Using a HMM Model for Gene Finding

Week - 13 presentation

By Ky, Aleks and Emil.

Objective of the project

- Investigate how to use a hidden Markov model for gene finding in prokaryotes.
- 2 Staphylococcus genomes, each containing several genes.
- The genomes are between 1.8 million and 2.8 million nucleotides
- Use the 7 and 3 state models to infer genome structure:
  C = coding
  N = non-coding
  R = reverse-coding

Objective of the project

- Two genomes, and two true annotations given

- Compare the calculated annotations with the true annotations and determine accuracy.

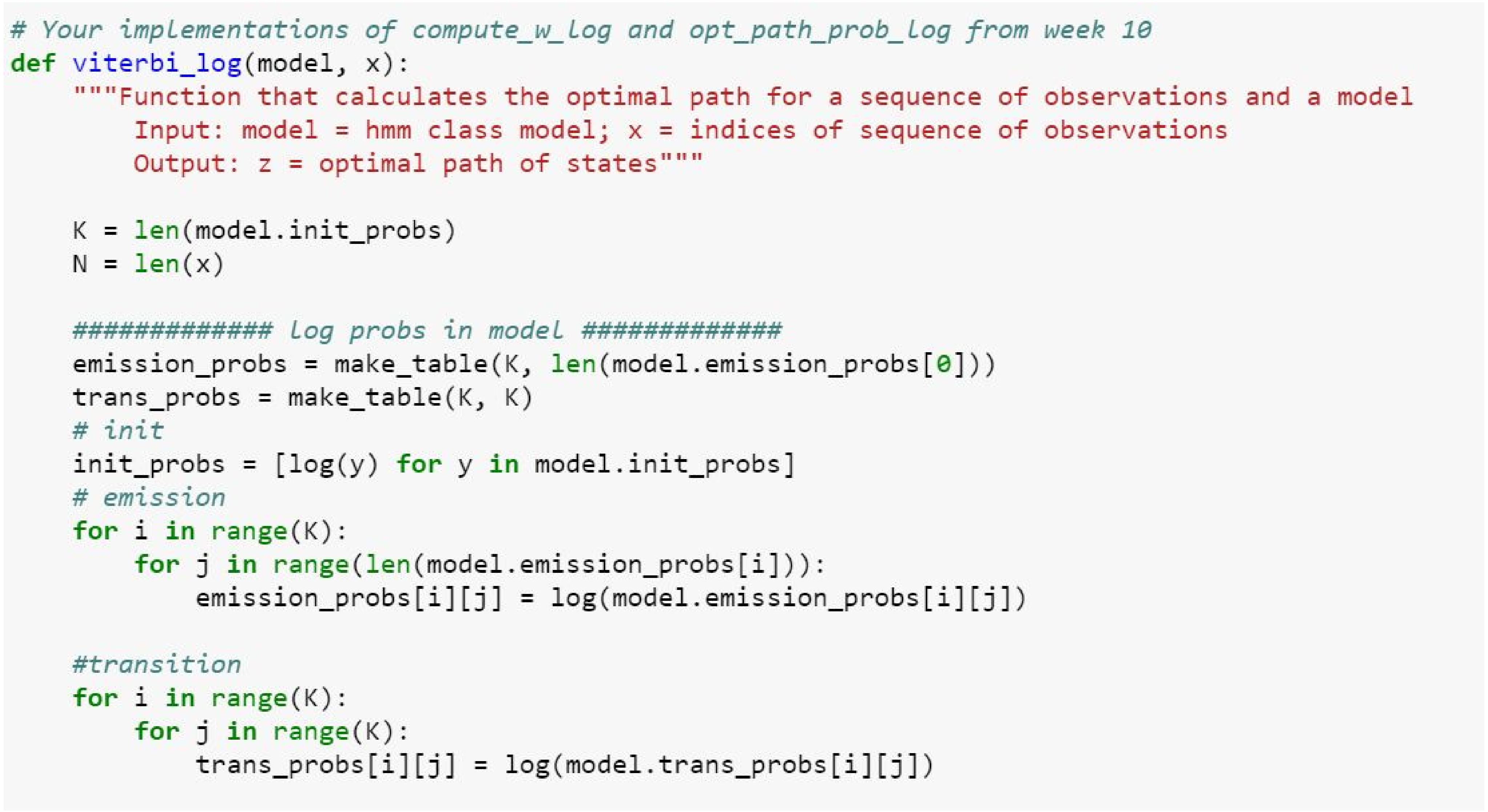
- Determine best way to implement the least time consuming algorithm

- Train the HMM model on one genome and determine accuracy with true annotation again.

Implementation: Path to indices

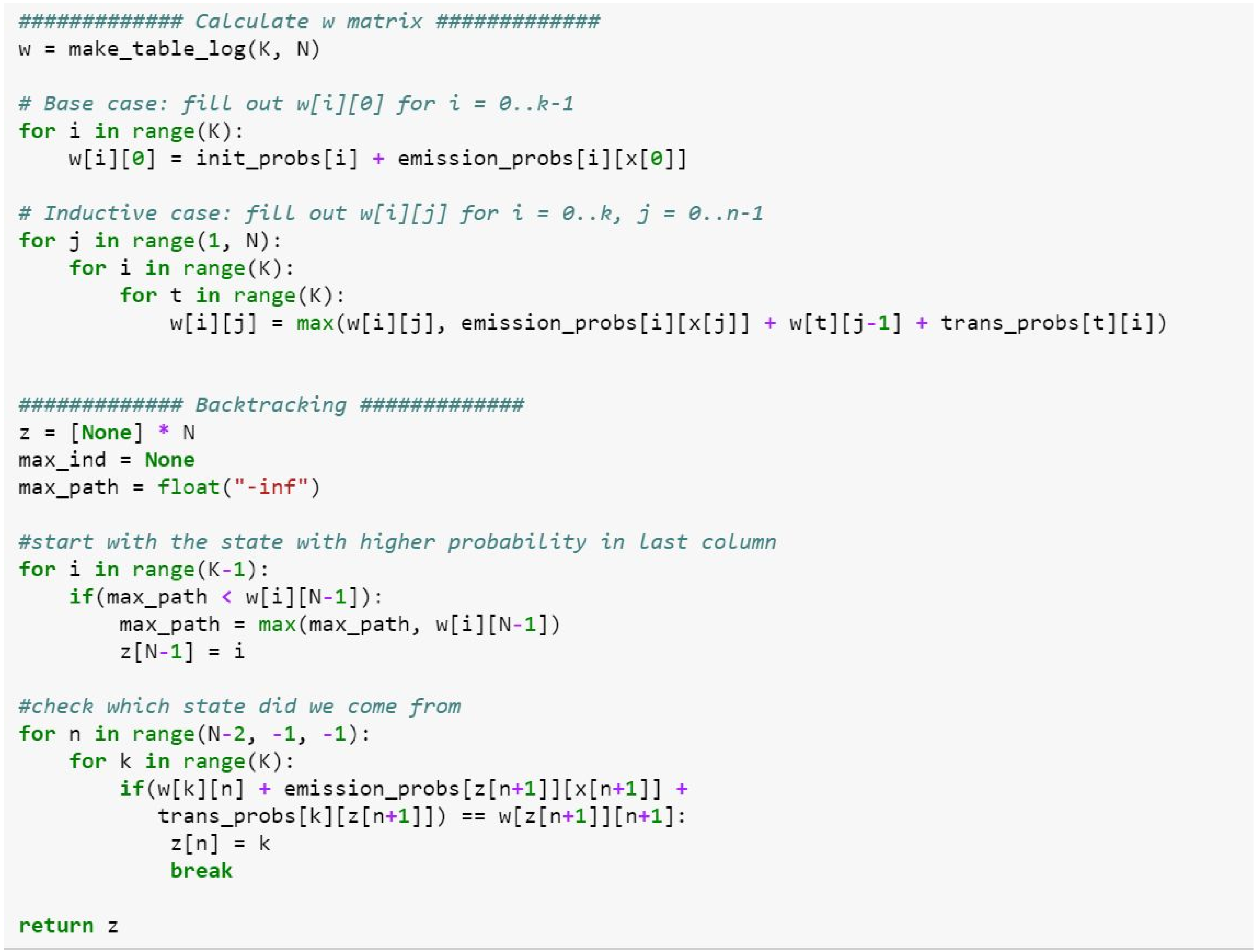
```python
def translate_path_to_indices_3state(obs):
    mapping = {"C":0, "c":0,"N":1, "n":1,"R": 2,"r":2}
    return [mapping[symbol.lower()] for symbol in obs]

def translate_path_to_indices_7state(obs):
    mapping = {"c":"012", "n":"3", "r":"456"}
    return [int(s) for s in [mapping[symbol.lower()] for symbol in obs]]

def translate_path_to_indices_7state(obs):
    lst1 = []
    c = 0
    r = 4

    for j in obs:
        if j is "N":
            lst1.append(3)
        if j is "C":
            lst1.append(c)
            if c != 2:
                c+=1
            else:
                c=0
        if j is "R":
            lst1.append(r)
            if r != 6:
                r+=1
            else:
                r=4

    return lst1
```

Viterbi log function:



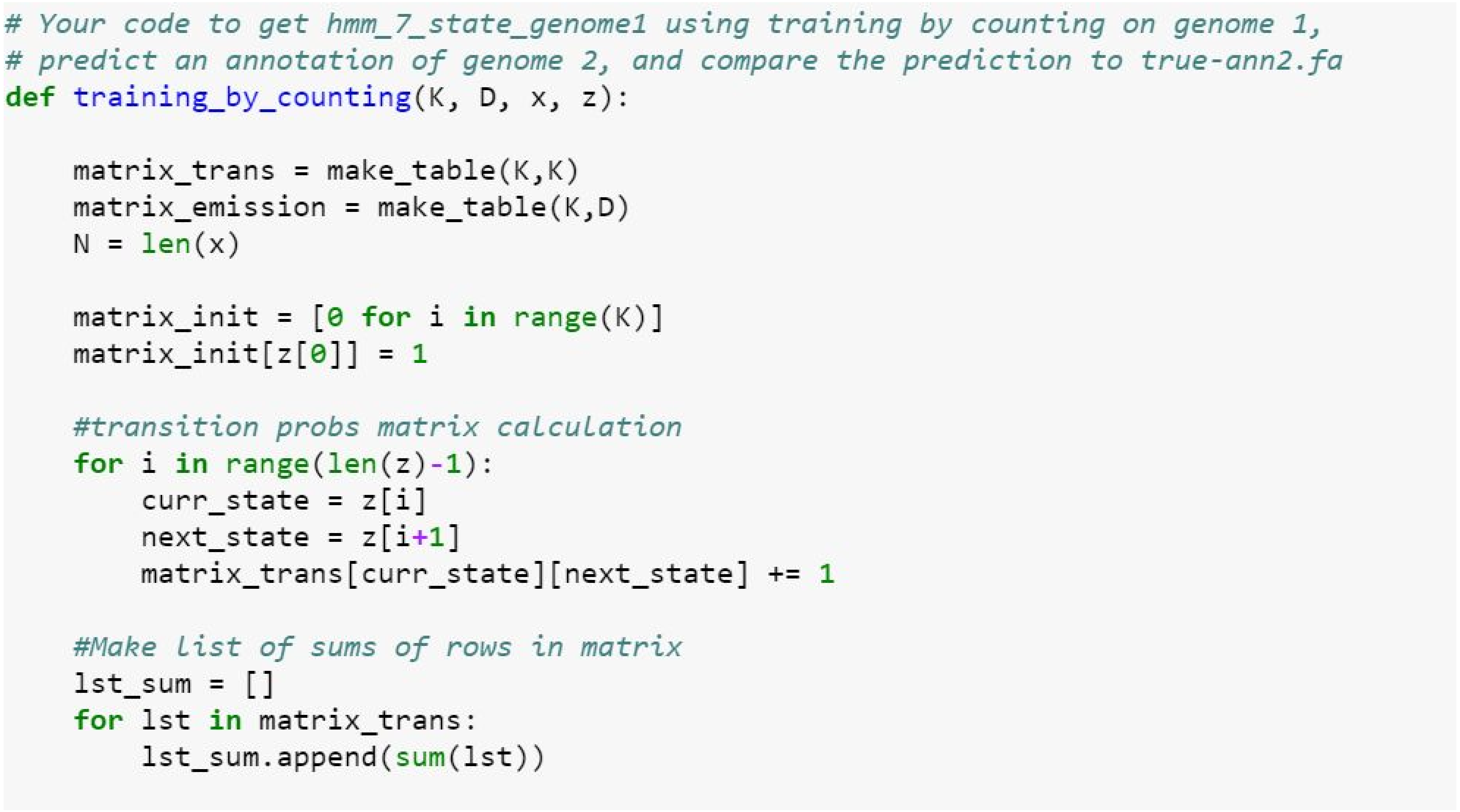
```python
# Your implementations of compute_w_log and opt_path_prob_log from week 10
def viterbi_log(model, x):
    """Function that calculates the optimal path for a sequence of observations and a model
       Input: model = hmm class model; x = indices of sequence of observations
       Output: z = optimal path of states"""

    K = len(model.init_probs)
    N = len(x)

    ############# log probs in model #############
    emission_probs = make_table(K, len(model.emission_probs[0]))
    trans_probs = make_table(K, K)
    # init
    init_probs = [log(y) for y in model.init_probs]
    # emission
    for i in range(K):
        for j in range(len(model.emission_probs[i])):
            emission_probs[i][j] = log(model.emission_probs[i][j])

    #transition
    for i in range(K):
        for j in range(K):
            trans_probs[i][j] = log(model.trans_probs[i][j])
```

Viterbi log function:



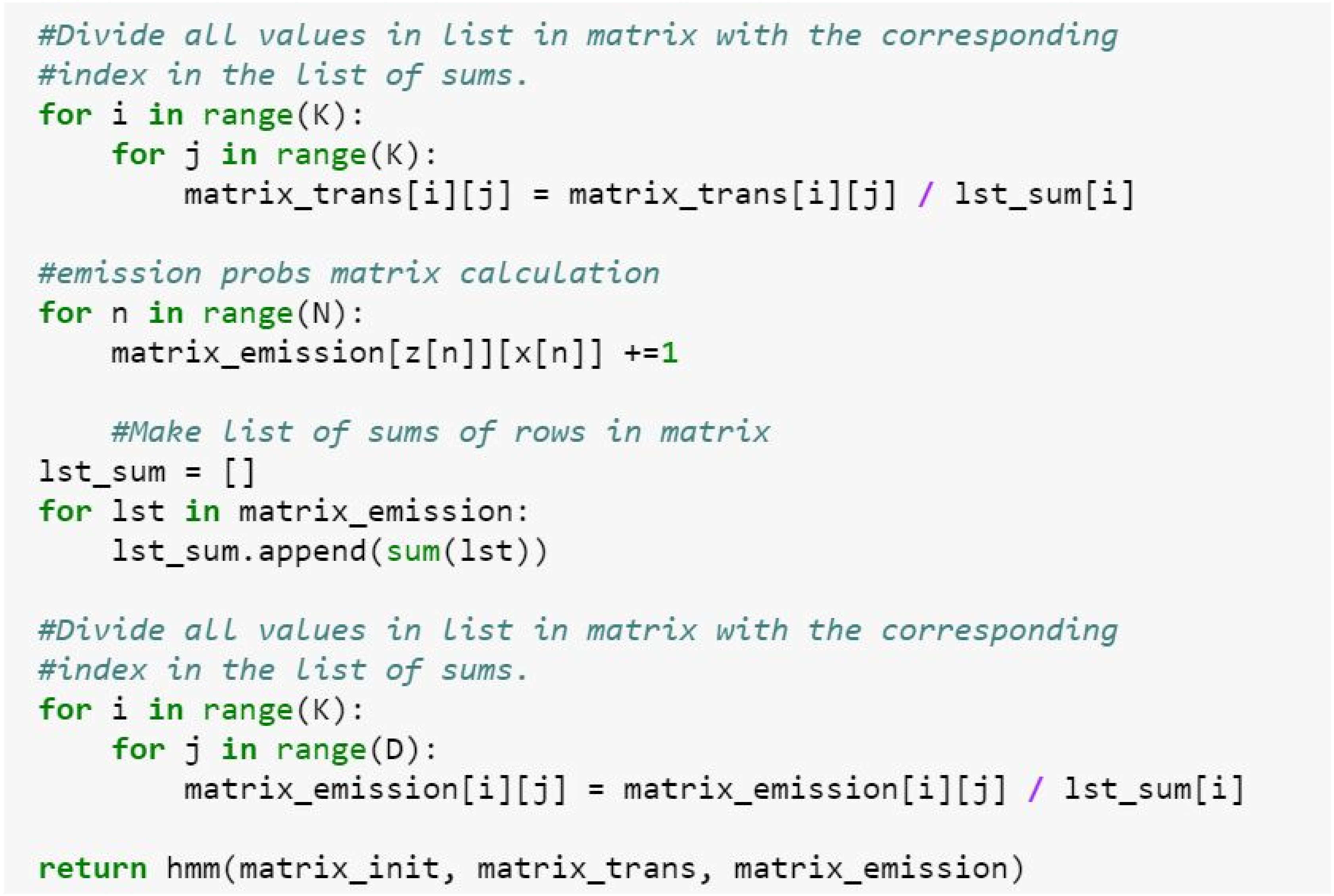
```python
############# Calculate w matrix #############
w = make_table_log(K, N)

# Base case: fill out w[i][0] for i = 0..k-1
for i in range(K):
    w[i][0] = init_probs[i] + emission_probs[i][x[0]]

# Inductive case: fill out w[i][j] for i = 0..k, j = 0..n-1
for j in range(1, N):
    for i in range(K):
        for t in range(K):
            w[i][j] = max(w[i][j], emission_probs[i][x[j]] + w[t][j-1] + trans_probs[t][i])


############# Backtracking #############
z = [None] * N
max_ind = None
max_path = float("-inf")

#start with the state with higher probability in last column
for i in range(K-1):
    if(max_path < w[i][N-1]):
        max_path = max(max_path, w[i][N-1])
        z[N-1] = i

#check which state did we come from
for n in range(N-2, -1, -1):
    for k in range(K):
        if(w[k][n] + emission_probs[z[n+1]][x[n+1]] +
           trans_probs[k][z[n+1]]) == w[z[n+1]][n+1]:
            z[n] = k
            break

return z
```

Training by counting



```python
# Your code to get hmm_7_state_genome1 using training by counting on genome 1,
# predict an annotation of genome 2, and compare the prediction to true-ann2.fa
def training_by_counting(K, D, x, z):

    matrix_trans = make_table(K,K)
    matrix_emission = make_table(K,D)
    N = len(x)

    matrix_init = [0 for i in range(K)]
    matrix_init[z[0]] = 1

    #transition probs matrix calculation
    for i in range(len(z)-1):
        curr_state = z[i]
        next_state = z[i+1]
        matrix_trans[curr_state][next_state] += 1

    #Make list of sums of rows in matrix
    lst_sum = []
    for lst in matrix_trans:
        lst_sum.append(sum(lst))
```

Training by counting



```python
#Divide all values in list in matrix with the corresponding
#index in the list of sums.
for i in range(K):
    for j in range(K):
        matrix_trans[i][j] = matrix_trans[i][j] / lst_sum[i]

#emission probs matrix calculation
for n in range(N):
    matrix_emission[z[n]][x[n]] +=1

    #Make list of sums of rows in matrix
lst_sum = []
for lst in matrix_emission:
    lst_sum.append(sum(lst))

#Divide all values in list in matrix with the corresponding
#index in the list of sums.
for i in range(K):
    for j in range(D):
        matrix_emission[i][j] = matrix_emission[i][j] / lst_sum[i]

return hmm(matrix_init, matrix_trans, matrix_emission)
```

Results - Given models

Accuracy (predicted vs true annotation)	3-state model	7-state model
Prediction on genome 1	31.87% Q1	39.19% Q2
Prediction on genome 2	35.09% Q3	37.19% Q4

Performance (elapsed time in seconds)	3-state model			7-state model		
	jupyter	Python 3.6	Python 2.7	jupyter	Python 3.6	Python 2.7
Prediction on genome 1	9.973	9.207	5.636	38.373	36.855	19.394
Prediction on genome 2	10.879	10.799	6.814	45.581	44.803	24.442

Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 8
Ubuntu 17.10 - GNU/Linux 4.13

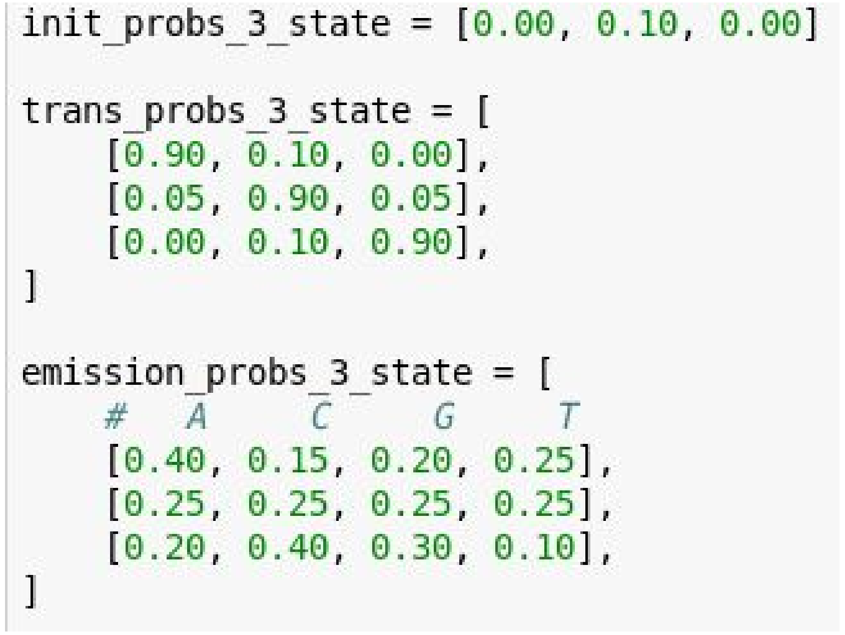
Results - Trained models

Accuracy (predicted vs true annotation)	3-state model		7-state model	
Prediction on genome 1	31.87%	Q1	39.19%	Q2
Prediction on genome 2	35.09%	Q3	37.19%	Q4

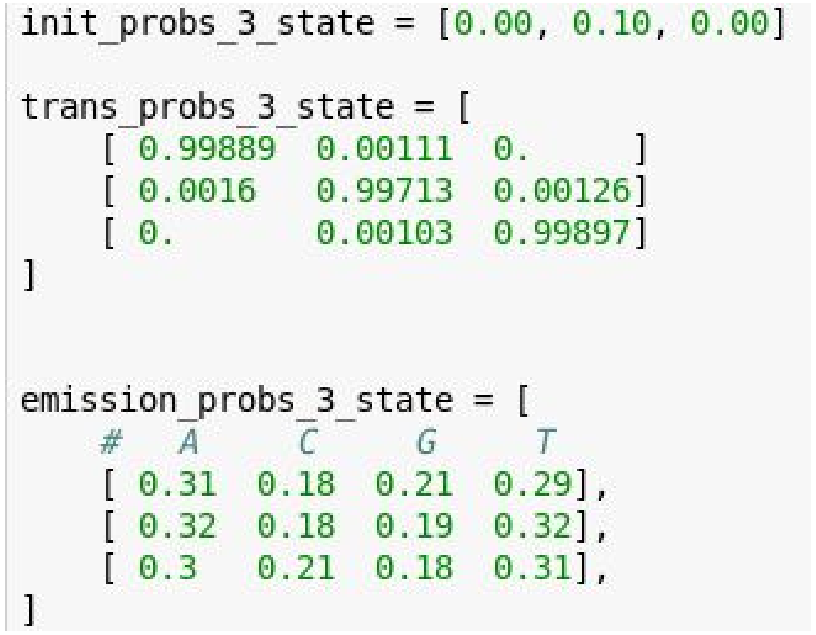
Accuracy (predicted vs true annotation)	3-state model	7-state model
Trained on genome 2 Prediction on genome 1	59.20% Q8	76.43% Q7
Trained on genome 1 Prediction on genome 2	57.37% Q6	78.30% Q5

Discussion

- Trained models > Given models



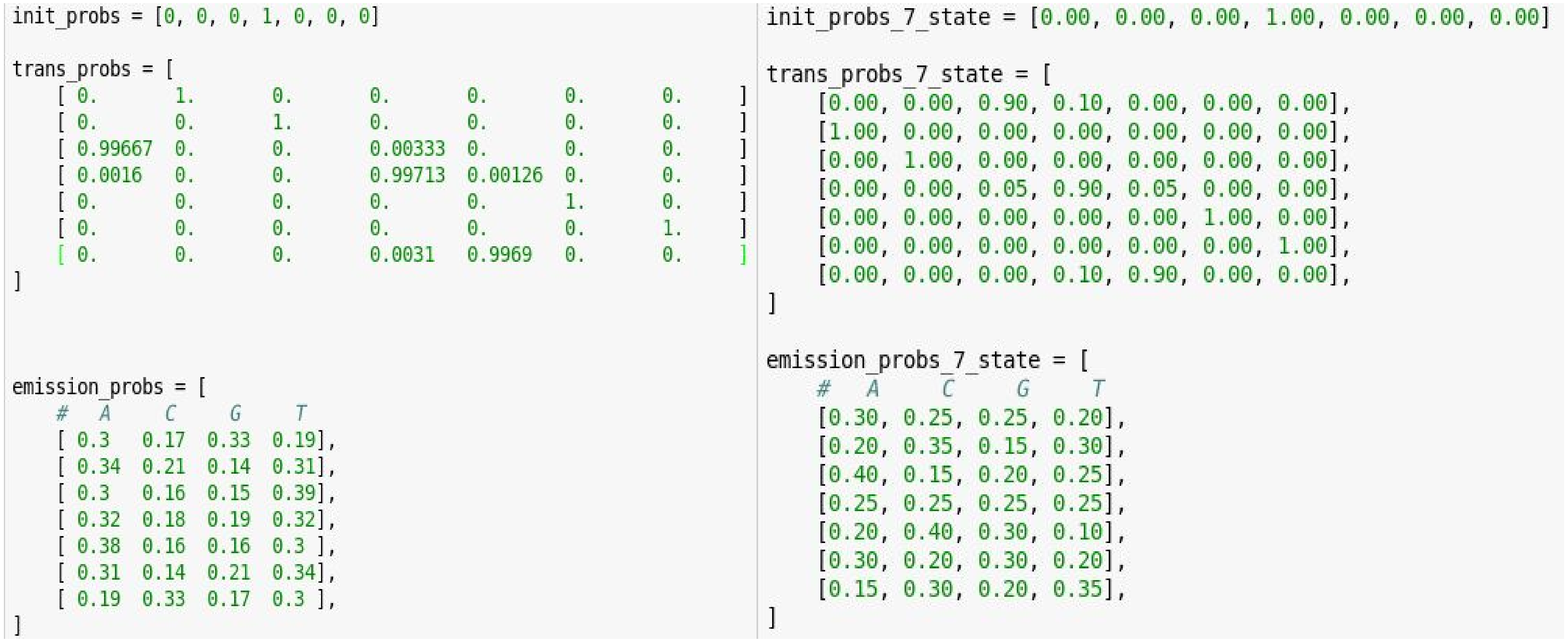
```
init_probs_3_state = [0.00, 0.10, 0.00]

trans_probs_3_state = [
    [0.90, 0.10, 0.00],
    [0.05, 0.90, 0.05],
    [0.00, 0.10, 0.90],
]

emission_probs_3_state = [
    #   A     C     G     T
    [0.40, 0.15, 0.20, 0.25],
    [0.25, 0.25, 0.25, 0.25],
    [0.20, 0.40, 0.30, 0.10],
]
```



```
init_probs_3_state = [0.00, 0.10, 0.00]

trans_probs_3_state = [
    [ 0.99889  0.00111  0.     ]
    [ 0.0016   0.99713  0.00126]
    [ 0.       0.00103  0.99897]
]


emission_probs_3_state = [
    #   A     C     G     T
    [ 0.31  0.18  0.21  0.29],
    [ 0.32  0.18  0.19  0.32],
    [ 0.3   0.21  0.18  0.31],
]
```

Discussion

- Trained models > Given models



```
init_probs = [0, 0, 0, 1, 0, 0, 0]

trans_probs = [
    [ 0.       1.       0.       0.       0.       0.       0.      ]
    [ 0.       0.       1.       0.       0.       0.       0.      ]
    [ 0.99667  0.       0.       0.00333  0.       0.       0.      ]
    [ 0.0016   0.       0.       0.99713  0.00126  0.       0.      ]
    [ 0.       0.       0.       0.       0.       1.       0.      ]
    [ 0.       0.       0.       0.       0.       0.       1.      ]
    [ 0.       0.       0.       0.0031   0.9969   0.       0.      ]
]


emission_probs = [
    #  A     C     G     T
    [ 0.3   0.17  0.33  0.19],
    [ 0.34  0.21  0.14  0.31],
    [ 0.3   0.16  0.15  0.39],
    [ 0.32  0.18  0.19  0.32],
    [ 0.38  0.16  0.16  0.3 ],
    [ 0.31  0.14  0.21  0.34],
    [ 0.19  0.33  0.17  0.3 ],
]
```

```
init_probs_7_state = [0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 0.00]

trans_probs_7_state = [
    [0.00, 0.00, 0.90, 0.10, 0.00, 0.00, 0.00],
    [1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00],
    [0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00],
    [0.00, 0.00, 0.05, 0.90, 0.05, 0.00, 0.00],
    [0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00],
    [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00],
    [0.00, 0.00, 0.00, 0.10, 0.90, 0.00, 0.00],
]

emission_probs_7_state = [
    #  A     C     G     T
    [0.30, 0.25, 0.25, 0.20],
    [0.20, 0.35, 0.15, 0.30],
    [0.40, 0.15, 0.20, 0.25],
    [0.25, 0.25, 0.25, 0.25],
    [0.20, 0.40, 0.30, 0.10],
    [0.30, 0.20, 0.30, 0.20],
    [0.15, 0.30, 0.20, 0.35],
]
```