Πανεπιστήμιο Πειραιώς

University of Piraeus

Εργασία Βιοπληροφορικής

Εαρινό εξάμηνο ακαδημαϊκού έτους 2022-2023

Μέλη:

Κούκη Τζανέτο, Π20097

Παπαλόης Γεώργιος, Π20222

Σπυροπούλου Κανέλλα, Π20179

Φραγκογιάννης Διονύσιος, Π20198

Περιεχόμενα

Εισαγωγή

Η γλώσσα που χρησιμοποιήθηκε ήταν η python. Τα προγράμματα εμφανίζουν τον τρόπο νίκης ή λύσης των προβλημάτων με την λογική του δυναμικού προγραμματισμού. Στα παιχνίδια, θεωρούμε ότι ο Παίκτης Α είναι ο παίκτης που παίζει πρώτος.

Άσκηση 11.4

Πίνακας μεταβάσεων

a	b
0.5	0.5

Πίνακας εκπομπής

	a	b
a	0.9	0.1
b	0.1	0.9

Πίνακας προτεραιότητας

	A	G	T	C
a	0.4	0.4	0.1	0.1
b	0.2	0.2	0.3	0.3

Στη συνέχεια λογαριθμίζουμε με βάση το 10.

a	b
-0.30103	-0.30103

	a	b
a	-0.045757	-1.00000
b	-1.00000	-0.045757

	A	G	T	C
a	-0,39794	-0,39794	-1.00000	-1.00000

b	-0.69897	-0.69897	-0.52287	-0.52287

Εξήγηση κώδικα

```
#Q = ονόματα καταστάσεων
Q = ['a','b']
#A = Ονόματα συμβόλων αλληλουχιών
A = ['A','G','T','C']
#N = Αριθμητική αναπαράσταση συμβόλων
N = [1,2,3,4]

#Τυπώνει τους πίνακες σχέσεων του HMM
def print_HMM(p,t,e):
    df = pd.DataFrame(p.tolist(), columns=['Start'], index=['State a',
'State b'])
    print(df)
    df = pd.DataFrame(t.tolist(), columns=['State a', 'State b'],
index=['State a', 'State b'])
    print(df)
    df = pd.DataFrame(e.tolist(), columns=['A', 'G', 'T', 'C'],
index=['State a', 'State b'])
    print(df)

#Μετατρέπει την αλληλουχία σε αριθμητική λίστα όπου
#A = 1 , G = 2 , T = 3 , C = 4
def sequence_str_to_numeric_list(sequence):
    list = []
    for i in range(len(sequence)):
        list.append(N[A.index(sequence[i])])
    return list

#Με όρισμα μια λίστα επιστρέφει πόσες φορές εμφανίζεται το max element
της λίστας
def count_max(prob):
    max_ = max(prob)
    times = prob.count(max_)
    if times > 1:
        return True
    else:
        return False

#Στην περίπτωση που υπάρχουν πολλές ισοπίθανες καταστάσεις με τη max
πιθανότητα
#επιστρέφει το index της κατάστασης που θα επιλεγεί σε αυτή τη
περίπτωση
def multiple_max_possibility(prob,trans,kat,observation):
    max_ = max(prob)
```

```
    index = -sys.maxsize - 1
    p = float('-inf')
    for rows in range(len(prob)):
        if prob[rows] == max_:
            if trans[kat][rows] > p:
                p = trans[kat][rows]
                index = rows
    return index

#Μετατρέπει τη τελική ακολουθία καταστάσεων από λίστα σε string
def states_list_to_str(states_path):
    str = ""
    for char in range(len(states_path)):
        if char == len(states_path)-1:
            str += states_path[char]
            break
        str = str + states_path[char] + " → "
    return str
```

Αλγόριθμος Viterbi

```
def viderbi_algorithm(pi,trans,emis,sequence):
    num_hid = trans.shape[0] # αριθμός κρυφών καταστάσεων
    num_obs = len(sequence)   # αριθμός παρατηρήσεων (όχι παρατήρηση
*καταστάσεων*)
    print("Number of hidden states",num_hid)
    print("Number of observations",num_obs)

    V = np.zeros((num_hid, num_obs)) # πίνακας για τις πιθανότητες
    for i in range(num_obs):
        for j in range(num_hid):
            if i == 0:
                V[j][i] = pi[j] + emis[j][sequence[i]-1]
            else:
                p = []
                for k in range(num_hid):
                    p.append(V[k][i-1]+trans[k][j])
                V[j][i] = emis[j][sequence[i]-1] + max(p)
    V = np.array(V)
    # Δημιουργία DataFrame για τον πίνακα V
    df = pd.DataFrame(V.tolist(), columns=[A[char-1] for char in
sequence], index=Q)
    print("\n",df,"\n")

    katastaseis = [] # η πιο πιθανή ακολουθία καταστάσεων
    for column in range(num_obs):
```

```
        probabilities = []
        for row in range(num_hid):
            probabilities.append(V[row][column])
        if count_max(probabilities):
            katastaseis.append(Q[multiple_max_possibility(probabilities
,trans,Q.index(katastaseis[column-1]),column)])
        else:
            katastaseis.append(Q[probabilities.index(max(probabilities)
)])
    return katastaseis,10**(max(probabilities))
```

Στο συγκεκριμένο κομμάτι, υλοποιούμε τον αλγόριθμο Viterbi, για ένα κερυμμένο μοντέλο Markov (HMM).

Ξεκινάμε με την αρχικοποίηση μεταβλητών. Στη συνέχεια, γίνεται ο υπολογισμός πιθανοτήτων για κάθε κατάσταση σε κάθε παρατήρηση. (Εάν είμαστε στην πρώτη παρατήρηση (i == 0), η πιθανότητα για μια κατάσταση j υπολογίζεται ως το άθροισμα του προηγούμενου προτεραιότητας pi[j] και της εκπομπής emis[j][sequence[i] - 1] στην παρατήρηση sequence[i].

Σε διαφορετική περίπτωση, δημιουργούμε μια λίστα p και υπολογίζουμε την πιθανότητα για κάθε πιθανή μετάβαση από την προηγούμενη παρατήρηση στην τρέχουσα παρατήρηση. Η τελική πιθανότητα για την κατάσταση j υπολογίζεται ως το άθροισμα της εκπομπής emis[j][sequence[i] - 1] και του μέγιστου στοιχείου της λίστας p.)

Δημιουργούμε DataFrame για τον πίνακα V. Τέλος, βρίσκουμε την πιο πιθανή ακολουθία καταστάσεων και επιστρέφουμε το αποτέλεσμα. (Αυτό το τμήμα του κώδικα υπολογίζει την πιο πιθανή ακολουθία καταστάσεων με βάση τις πιθανότητες στον πίνακα V. Για κάθε στήλη του V, υπολογίζονται οι πιθανότητες για κάθε κατάσταση και ελέγχεται αν υπάρχει περισσότερη από μία καταστάσεις με τη μέγιστη πιθανότητα. Στην περίπτωση αυτή, χρησιμοποιείται η συνάρτηση multiple_max_possibility για να επιλεγεί η πιο πιθανή κατάσταση με βάση την προηγούμενη κατάσταση. Σε διαφορετική περίπτωση, επιλέγεται η κατάσταση με τη μεγαλύτερη πιθανότητα.)

Συνέχεια κώδικα

```
priors = np.array([0.5, 0.5])
transition = np.array([[0.9, 0.1],
                       [0.1, 0.9]])
emission = np.array([[0.4, 0.4, 0.1,0.1],
                     [0.2, 0.2, 0.3,0.3]])
```

Δημιουργούμε τους πίνακες προτεραιότητας, μετάβασης και εκπομπής.

```
print("The HMM with array representation:")
print_HMM(priors,transition,emission)
```

Εμφανίζουμε το HMM με χρήση απλών πιθανοτήτων.

```
#Μετατροπή πιθανοτήτων σε λογαριθμικές με βάση το 10
priors = np.log10(priors)
transition = np.log10(transition)
emission = np.log10(emission)

print("\nLogarithmic probabilities with base 10\n")
print_HMM(priors,transition,emission)
```

```
seq = "GGCT"
s,p =
viderbi_algorithm(priors,transition,emission,sequence_str_to_numeric_li
st(seq))
print("The most probable path is:",states_list_to_str(s))
print("Its probability is (we used log10):",p)
```

Εκτελούμε τον αλγόριθμο Viterbi για την ακολουθία παρατηρήσεων "GGCT" και τυπώνουμε την πιο πιθανή ακολουθία.

Output

```
The HMM with array representation:
State b -0.69897 -0.69897 -0.522879 -0.522879
Number of hidden states 2
Number of observations 4

          G         G         C         T
a -0.69897 -1.142668 -2.188425 -3.234182
b -1.00000 -1.744727 -2.313364 -2.882000

The most probable path is: a → a → a → b
Its probability is (we used log10): 0.0013121999999999988
```

Άσκηση 6.12

Δυναμική επίλυση του προβλήματος

Το συγκεκριμένο παιχνίδι παίζεται με 2 παίκτες και 2 αλληλουχίες νουκλεοτιδίων. Κάθε παίκτης διαγράφει την μία ακολουθία και χωρίζει την άλλη σε 2 κομμάτια. Νικητής είναι αυτός που διαγράφει το τελευταίο νουκλεοτίδιο.

Πιο συγκεκριμένα, αν έχουμε μια από τις δύο αλληλουχίες με μήκος 1, αρκεί να επιλέξουμε εκείνη για να κερδίσουμε.

Με μια αλληλουχία μήκους 2, μπορούμε να την χωρίσουμε σε 1 και 1. Από το προηγούμενο συμπέρασμα, ο επόμενος παίκτης κερδίζει άρα εμείς χάνουμε.

Για μήκος 3 χωρίζουμε σε 1 και 2. Αν ο επόμενος παίκτης επιλέξει την ακολουθία 2, ξέρουμε ότι θα χάσει, ενώ αν επιλέξει την 1 θα κερδίσει. Επειδή και οι 2 παίκτες επιλέγουν την

καλύτερη στρατηγική, ο επόμενος παίκτης θα επιλέξει την αλληλουχία που κερδίζει, άρα ο παίκτης Α θα χάσει.

Για μήκος 4, ο παίκτης Α έχει την επιλογή να χωρίσει σε 1 και 3 ή 2 και 2. Αν χωρίσει σε 1 και 3, από τα προηγούμενα συμπεράσματα ξέρουμε ότι θα χάσει, ενώ αν επιλέξει το 2 και 2 θα κερδίσει. Ο παίκτης Α θα επιλέξει το 2 και 2 άρα θα κερδίσει.

Αρκεί να συνεχίσουμε την διαδικασία για την αλληλουχία με μεγαλύτερο μήκος.

Εξήγηση κώδικα

```
def fillSeq(selectedSeqName):
    seqFile = open(" ./auxiliary2023/" + selectedSeqName + ".txt", "r")

    seq = []
    lines = seqFile.readlines()[1:]
    for line in lines:
        for letter in line:
            seq.append(letter)
    seqFile.close()
    return seq

seq1 = fillSeq(input("Select first sequence to use (brain, liver or
muscle)"))
seq2 = fillSeq(input("Select second sequence to use (brain, liver or
muscle)"))
```

Διαβάζουμε και να αποθηκεύουμε τις δύο αλληλουχίες της επιλογής μας.

```
if len(seq2) > len(seq1):
    temp = seq1
    seq1 = seq2
    seq2 = temp
```

Τοποθετούμε την μεγαλύτερη αλληλουχία στην λίστα seq1 και την μικρότερη στην seq2.

```
canWin = []
winningRemoval = []
canWin.append(False)
canWin.append(True)
```

Αρχικοποιούμε τις λίστες canWin και winningRemoval για να αποθηκεύσουμε ποια αλληλουχία κερδίζει και πως πρέπει να χωριστούν για να κερδίσουν.

Τοποθετήσαμε στην θέση 0 το False, γιατί αν η αλληλουχία έχει 0 στοιχεία έχουμε ήδη χάσει και στην θέση 1 το True γιατί ο σκοπός του παιχνιδιού είναι να αφαιρέσουμε το τελευταίο στοιχείο.

```
for i in range(2,len(seq1)+1):
    for j in range(1,int((i/2)+1)):
```

Πρέπει να εξετάσουμε όλες τις περιπτώσεις ξεκινώντας από το μήκος 2 μέχρι το μήκος της μεγαλύτερης ακολουθίας. Επιπλέον πρέπει να εξετάσουμε όλους τους πιθανούς χωρισμούς της αλληλουχίας. Πχ για 6, πρέπει να εξεταστούν οι περιπτώσεις [1,5], [2,4], [3,3].

```
if not(canWin[j]) and not(canWin[i-j]):
            canWin.append(True)
            winningSplit = []
            winningSplit.append(j)
            winningSplit.append(i-j)
            winningRemoval.append(winningSplit)
            break
```

Στην συνέχεια, ελέγχουμε και οι δύο αλληλουχίες σε κάθε περίπτωση επιστρέφουν False. Αν επιστρέφουν False, σημαίνει ότι ο επόμενος παίκτης θα χάσει ότι και να επιλέξει, άρα εμείς θα κερδίσουμε. Αν, λοιπόν, συμβεί αυτό, αποθηκεύουμε στην λίστα winningRemoval το σημείο που χωρίστηκε. Επιπλέον, ενημερώνουμε στην λίστα canWin ότι στο σημείο αυτό ο παίκτης μπορεί να κερδίσει.

```
 else:
        canWin.append(False)
```

Αν ολοκληρωθεί το εμφωλευμένο loop χωρίς να συναντήσει τον «χωρισμό» που κερδίζει, η θα τοποθετηθεί στην λίστα canWin False.

```
if (canWin[-1] or canWin[len(seq2)]):
    print("Player A won.")
else:
    print("Player A lost.")
```

Όταν τελειώσει η επανάληψη, ελέγχουμε τα στοιχεία στις θέσεις μήκους των δύο αλληλουχιών. Αν κάποια από τις δύο είναι True, ο παίκτης Α έχει κερδίσει.

```
print("Winning splits were:")
for i in winningRemoval:
    print(i)
```

Τέλος, εμφανίζουμε τους χωρισμούς που επιτρέπουν να κερδίσουν οι παίκτες.

Output

```
Select first sequence to use (brain, liver or muscle)muscle
Select second sequence to use (brain, liver or muscle)brain
Player A won.
Winning splits were:
[2, 2]
[2, 3]
[3, 3]
[2, 7]
[2, 8]
```

Νικηφόρα στρατηγική

Από το output συμπεραίνουμε ότι για να κερδίσει ένας παίκτης αρκεί να χωρίζει σε την αλληλουχία με τρόπο έτσι ώστε ένα από τα δύο μέρη να έχει μήκος 2 ή 3.

Άσκηση 6.14

Δυναμική επίλυση του προβλήματος

Το συγκεκριμένο παιχνίδι παίζεται με 2 παίκτες και 2 αλληλουχίες νουκλεοτιδίων. Κάθε παίκτης διαγράφει ένα στοιχείο από την μία αλληλουχία και 2 από την άλλη. Νικητής είναι αυτός που δεν μπορεί να κάνει άλλη κίνηση.

Για την επίλυση αυτού του προβλήματος αρκεί να ξεκινήσουμε από την κατάσταση [0,0] και να βρίσκουμε τις επόμενες σύμφωνα με τις προηγούμενες.

Αν ο παίκτης Α ξεκινήσει με 2 αλληλουχίες μήκους 0, δεν μπορεί να κάνει κάποια κίνηση άρα κερδίζει. Επομένως, για [0,0] ο παίκτης κερδίζει.

Για τις αλληλουχίες [0,1] επίσης κερδίζει.

Για τις αλληλουχίες [1,3] μπορεί να αφαιρέσει ένα από την πρώτη και 2 από την δεύτερη. Άρα ο επόμενος παίκτης θα έχει την ακολουθία [0,1], οπού ξέρουμε ότι κερδίζει από το προηγούμενο συμπέρασμα, άρα ο παίκτης Α χάνει. Υπολογίζουμε με τον ίδιο τρόπο για [n, m].

Εξήγηση κώδικα

```
def fillSeq(selectedSeqName):
    seqFile = open("./auxiliary2023/" + selectedSeqName + ".txt", "r")
    seq = []
    lines = seqFile.readlines()[1:]
    for line in lines:
        for letter in line:
            seq.append(letter)
    seqFile.close()
    return seq

seq1 = fillSeq(input("Select first sequence to use (brain, liver or
muscle)"))
seq2 = fillSeq(input("Select second sequence to use (brain, liver or
muscle)"))
```

Αποθησκεύουμε τις αλληλουχίες.

```
canWin = np.ones((len(seq1)+1, len(seq2)+1), dtype=bool)
wasRemovedFromI = np.zeros((len(seq1) + 1, len(seq2) + 1), dtype=int)
wasRemovedFromJ = np.zeros((len(seq1) + 1, len(seq2) + 1), dtype=int)
```

Δημιουργήθηκε το array canWin που περιέχει True αν ο παίκτης κερδίζει με τα συγκεκριμένα μήκη αλληλουχιών και False για όταν δεν κερδίζει. Αρχικοποιήθηκε με True.

Επιπλέον τα array wasRemovedFromI και wasRemovedFromJ για να αποθηκεύουμε αν από την πρώτη ή την δεύτερη αντίστοιχα αλληλουχία αφαιρέσαμε το 1 ή το 2.

```
for i in range(len(seq1)+1):
    for j in range(len(seq2)+1):
        if (i-1>=0 and j-2>=0) or (i>=2 and j>=1):
```

Ξεκινώντας από το 0 μέχρι το μήκος των αλληλουχιών ελέγχουμε αν μπορούμε να αφαιρέσουμε στοιχεία.

```
 else:
    canWin[i][j] = True
```

Αν δεν μπορούμε να αφαιρέσουμε σημαίνει ότι ο παίκτης κέρδισε.

```
if i-1>=0 and j-2>=0:
```

Αν μπορούμε να αφαιρέσουμε, ελέγχουμε αν γίνεται να αφαιρέσουμε 1 από την πρώτη και 2 από την δεύτερη,

```
if canWin[i-1][j-2]:
```

και ελέγχουμε αν ο επόμενος παίκτης μπορεί να κερδίσει.

```
canWin[i][j] = False
else:
       canWin[i][j] = True
     wasRemovedFromI[i][j] = -1
      wasRemovedFromJ[i][j] = -2
```

Αν μπορεί να κερδίσει, σημαίνει ότι ο παίκτης Α χάνει, αλλιώς κερδίζει. Επιπλέον ενημερώνουμε τα array wasRemovedFromI και wasRemovedFromJ.

```
if i>=2 and j>=1:
                if not(canWin[i][j]):
                    if canWin[i-2][j-1]:
                        canWin[i][j] = False
                    else:
                        canWin[i][j] = True
                    wasRemovedFromI[i][j] = -2
                    wasRemovedFromJ[i][j] = -1
```

Μετά ελέγχουμε για την αντίθετη περίπτωση. Δηλαδή αν αφαιρέσουμε από την πρώτη 2 και από την δεύτερη ένα.

```
if canWin[-1][-1]:
```

```
    print("Player A won.")
else:
    print("Player A lost.")
```
Αν το τελευταίο στοιχείο του array canWin είναι True, ο παίκτης Α κέρδισε.

```
for i in range(len(seq1)+1):
    for j in range(len(seq2)+1):
        print("["+str(i)+", "+str(j)+"]" +str(canWin[i][j]))
        print("["+str(wasRemovedFromI[i][j])+",
"+str(wasRemovedFromJ[i][j])+"]")
```
Τέλος εμφανίζουμε με ποιες αφαιρέσεις κερδίζει ο παίκτης Α.

Output

```
C:\Users\kuqit\AppData\Local\Programs\Python\Python39\python.exe "C
Select first sequence to use (brain, liver or muscle)brain
Select second sequence to use (brain, liver or muscle)liver
50656 seq1 length
39875 seq2 length
```

Νικηφόρα στρατηγική

Παρατηρούμε ότι για να κερδίσει ένας παίκτης, έχοντας δύο αλληλουχίες μήκους n και m αντίστοιχα, για m μεγαλύτερο του 2, αν το n είναι περιττός αριθμός πρέπει να αφαιρέσει από το n 2 και από το m 1, ενώ για άρτιο μήκος n το αντίστροφο. Αν το m είναι 0, ο παίκτης Α κερδίζει επειδή δεν μπορεί να κάνει κάποια κίνηση. Για m ίσο με 2, αν το n είναι άρτιος κερδίζει με την παραπάνω λογική και για m ίσο με 1, χάνει πάντα, εκτός από την περίπτωση που το n είναι μικρότερο του 2, αφού δεν μπορεί να κάνει κάποια άλλη κίνηση.