Question 4

a. p0: [ 0.88 0. 0. 0. 0.04 0. 0.09 0. ]

b. First 5\*5 values

Tr: [[ 6.02e-01 3.41e-02 1.21e-01 3.68e-02 1.03e-01]

[ 4.61e-01 2.19e-02 4.18e-02 2.53e-02 4.52e-02]

[ 8.54e-02 3.56e-03 8.15e-01 4.90e-03 5.42e-03]

[ 1.61e-01 7.77e-03 2.13e-02 7.03e-01 3.13e-02]

[ 5.62e-02 5.89e-04 7.56e-04 3.39e-03 9.18e-01]]

Stationary Distribution: [ 1.78e-01 1.17e-02 2.24e-01 3.89e-02

3.36e-01 9.79e-05 2.10e-01 3.17e-04]

c. Ob: [[ 0.06 0.01 0.07 0.05 0.03]

[ 0.05 0.02 0.05 0.04 0.05]

[ 0.06 0.02 0.03 0.05 0.06]

[ 0.09 0.01 0.08 0.09 0.04]

[ 0.12 0.01 0.05 0.09 0.04]]

d. We can take one more random variable for size. Also another approach could to create a random variable which has the probability of any character being a delimiter.

e. I used a small a model, the one used in the class to check the code, and manually confirmed the output

File:1l9g.txt P(x6|o(0)) = [ 7.39e-02 4.83e-03 1.59e-01 3.39e-02

6.79e-01 1.62e-05 4.85e-02 1.47e-04]

File:1h6h.txt P(x9|o(2)) = [ 1.26e-01 5.39e-03 2.71e-02 5.85e-02

6.77e-01 1.93e-05 1.05e-01 1.01e-04]

[File:1rdr.txt](file:///1rdr.txt) logp(o(4)) = -929.182828473

Code:

import numpy as np

from os import walk

mypath = 'proteins/' # use path to data files

\_, \_, filenames = next(walk(mypath), (None, None, []))

np.set\_printoptions(precision=2)

mSeq = len(filenames) # read in each sequence

# mSeq = 10

o,x = [],[]

for i in range(mSeq):

f = open('proteins/' + filenames[i] , 'r')

o.append( f.readline()[:-1] ) # strip trailing '\n'

x.append( f.readline()[:-1] )

f.close()

xvals, ovals = set(),set() # extract the symbols used in x and o

for i in range(mSeq):

xvals |= set(x[i])

ovals |= set(o[i])

xvals = list( np.sort( list(xvals) ) )

ovals = list( np.sort( list(ovals) ) )

dx,do = len(xvals),len(ovals)

for i in range(mSeq): # and convert to numeric indices

x[i] = np.array([xvals.index(s) for s in x[i]])

o[i] = np.array([ovals.index(s) for s in o[i]])

p0 = np.zeros(dx)

for i in range(mSeq):

p0[x[i][0]] += 1

p0 = p0/sum(p0)

print 'p0:', p0

Tr = np.zeros((dx,dx))

for seq in range(mSeq):

for s in range(len(x[seq])-1):

Tr[x[seq][s]][x[seq][s+1]] += 1

Tr = Tr/Tr.sum(axis=1)[:,None]

print 'Tr:', Tr[:5,:5]

print np.matmul(p0,np.linalg.matrix\_power(Tr,100))

Ob = np.zeros((dx,do))

for seq in range(mSeq):

for s in range(len(x[seq])):

Ob[x[seq][s]][o[seq][s]]+=1

Ob = np.asarray(Ob/Ob.sum(axis=1)[:,None])

print 'Ob:', Ob[:5,:5]

# o = [1,2,3]

def markovMarginals(o,p0,Tr,Ob):

'''Compute p(o) and the marginal probabilities p(x\_t|o) for a Markov model

defined by P[xt=j|xt-1=i] = Tr(i,j) and P[ot=k|xt=i] = Ob(i,k) as numpy matrices'''

dx,do = Ob.shape # if a numpy matrix

L = len(o)

f = np.zeros((L,dx))

r = np.zeros((L,dx))

p = np.zeros((L,dx))

f[0,:] = p0\*Ob[:,o[0]] # compute initial forward message

log\_pO = np.log(f[0,:].sum()) # update probability of sequence so far

f[0,:] /= f[0,:].sum() # normalize (to match definition of f)

for t in range(1,L): # compute forward messages

f[t,:] = np.matmul(f[t-1,:],Tr)\*Ob[:,o[t]]

log\_pO += np.log(f[t,:].sum())

f[t,:] /= f[t,:].sum()

r[L-1,:] = np.ones(dx) # initialize reverse messages

p[L-1,:] = r[L-1,:]\*f[L-1,:] # and marginals

for t in range(L-2,-1,-1):

r[t,:] = np.matmul(Tr,r[t+1,:]\*Ob[:,o[t+1]])

r[t,:] /= r[t,:].sum()

p[t,:] = r[t,:]\*f[t,:]

p[t,:] /= p[t,:].sum()

return log\_pO, p

def testMarkovMarginals():

Tr = np.asarray([[0, 0, 1],[.33, .66, 0], [.5, .5, 0]])

Ob = np.asarray([[1,0],[.5,.5],[0,1]])

p0 = np.asarray([.33, .33, .33])

O = [0,1]

log\_pO, p = markovMarginals(O,p0, Tr,Ob)

print p

testMarkovMarginals()

\_, p = markovMarginals(o[0],p0,Tr,Ob)

print p[6]

\_, p = markovMarginals(o[2],p0,Tr,Ob)

print p[6]

log\_pO, \_ = markovMarginals(o[4],p0,Tr,Ob)

print log\_pO