Programming Hidden Markov Models (60 P)

In this exercise, you will experiment with hidden Markov models, in particular, applying them to modeling character sequences, and analyzing the learned solution. As a starting point, you are provided in the file `hmm.py` with a basic implementation of an HMM and of the Baum-Welch training algorithm. The names of variables used in the code and the references to equations are taken from the HMM paper by Rabiner et al. downloable from ISIS. In addition to the variables described in this paper, we use two additional variables: Z for the emission probabilities of observations O, and ψ (i.e. psi) for collecting the statistics of Equation (40c).

##Question 1: Analysis of a small HMM (30 P)

We first look at a toy example of an HMM trained on a binary sequence. The training procedure below consists of 100 iterations of the Baum-Welch procedure. It runs the HMM learning algorithm for some toy binary data and prints the parameters learned by the HMM (i.e. matrices A and B).

###Question 1a: Qualitative Analysis (15 P)

- *Run* the code several times to check that the behavior is consistent.
- *Describe* qualitatively the solution A, B learned by the model.
- *Explain* how the solution $\lambda = (A, B)$ relates to the sequence of observations O that has been modeled.

The result is consistent up to isomorphisims. Most of the time A, B and Pi are exactly the same (with permuted state orders).

Most of the time, A describes a model that has a deterministic state workflow (i.e. for each state there is exactly one state which is guarenteed to follow). Most of the time, B describes the same symbol probabilities (pairs are 0.8/0.2, 0/1, 0.88/0.12 and 0.72/0.28). It is also worth noting that these probabilities are indeed "executed" in the same order (most of the time) as enforced by A and Pi. Most of the time Pi desribes a model that deterministically selects one starting state (probability is mostly equal to one for a single state).

The solution can be related to the input sequence as follows: The resulting model simply cycles through each state and counts the number of times it encounters a "1" or "0" at a certain index modulo 4 (the state count). For example O[4k] = "1" for all k = 0,..,49 so the resulting symbol probability is 1 for "1" and 0 for "0". On the other hand we have 14 occurrences for which O[4k + 1] = "1" and 36 for which O[4k + 1] = "0" so this amounts to a 0.28 probabilty for encountering "1" and conversely a 0.72 probablity for a "0". Pi simply chooses the state which corresponds to the 1/0 distribution (naming does not matter).

In [74]:
```
o1 = (O[(numpy.arange(O.size)) % 4 == 0])
o2 = (O[(numpy.arange(O.size)) % 4 == 1])
o3 = (O[(numpy.arange(O.size)) % 4 == 2])
o4 = (O[(numpy.arange(O.size)) % 4 == 3])

print ("O[k]     -> ", numpy.sum(o1) / o1.size)
print ("O[k + 1] -> ", numpy.sum(o2) / o2.size)
print ("O[k + 2] -> ", numpy.sum(o3) / o3.size)
print ("O[k + 3] -> ", numpy.sum(o4) / o4.size)
```

```
O[k]     ->  1.0
O[k + 1] ->  0.28
O[k + 2] ->  0.2
O[k + 3] ->  0.12
```

In [4]:
```
import numpy,hmm

O = numpy.array([1,0,1,0,1,1,0,0,1,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,1,0,0,1,1,
                 0,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,
                 0,0,1,0,1,0,1,0,0,0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,
                 0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,
                 1,0,0,0,1,1,0,0,1,0,1,1,1,0,0,1,1,0,0,0,1,1,0,0,1,1,0,0,1,
                 0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,
                 0,0,1,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,0,0,0,1,1,0,0])

hmmtoy = hmm.HMM(4,2)

for k in range(100):
    hmmtoy.loaddata(O)
    hmmtoy.forward()
    hmmtoy.backward()
    hmmtoy.learn()

print('A')
print("\n".join([" ".join(['%.3f'%a for a in aa]) for aa in hmmtoy.A]))
print(' ')
print('B')
print("\n".join([" ".join(['%.3f'%b for b in bb]) for bb in hmmtoy.B]))
print(' ')
print('Pi')
print("\n".join(['%.3f'%b for b in hmmtoy.Pi]))
```

```
A
0.000 0.000 0.000 1.000
0.000 0.000 0.000 1.000
0.000 0.000 0.000 1.000
0.274 0.058 0.667 0.000

B
0.203 0.797
0.000 1.000
0.522 0.478
0.800 0.200

Pi
0.000
1.000
0.000
0.000
```

###Question 1b: Finding the best number N of hidden states (15 P)

For the same sequence of observations as in Question 1a, we would like to determine automatically what is a good number of hidden states $N = \mathrm{card}(S)$ for the model.

- *Split* the sequence of observations into a training and test set (you can assume stationarity).
- *Train* the model on the training set for several iteration (e.g. 100 iterations) and for multiple parameter N.
- *Show* for each choice of parameter N the log-probability $\log p(O|\lambda)$ for the test set. (If the results are unstable, perform several trials of the same experiment for each parameter N.)
- *Explain* in the light of this experiment what is the best parameter N.

In [121]:
```
def trainCardS(O, it = 100, NMax = 20, trials = 4, splitFac = 0.5):

    O_train = O[numpy.arange(numpy.round(O.size * splitFac).astype(int))]
    O_test  = O[numpy.arange(numpy.round(O.size * splitFac).astype(int), O.size)]

    print ("Train size: ", O_train.size)
    print ("Test size: ", O_test.size)

    testLogErr = numpy.zeros(NMax);

    for N in range(1, NMax + 1):

        print ("\nN = ", N)

        testTrialErrs = numpy.zeros(trials)

        for t in range(1, trials + 1):

            HMM = hmm.HMM(N,2)

            for k in range(it):

                HMM.loaddata(O_train)
                HMM.forward()
                HMM.backward()
                HMM.learn()

            eLogTrain = numpy.log(HMM.pobs)

            HMM.loaddata(O_test)
            HMM.forward()

            eLogTest = numpy.log(HMM.pobs)

            testTrialErrs[t - 1] = eLogTest

            print ("trial ", t, " >> TrainLogP: ", eLogTrain, ", TestLogP: ", eLogTest)


        testLogErr[N - 1] = numpy.mean(testTrialErrs)
        print ("-> Average TestErrorLog: ", testLogErr[N - 1])

    return  testLogErr.argmin() + 1

print ("\nBest N: ", trainCardS(O, NMax = 20, trials = 10))
```

```
Train size:  100
Test size:  100

N =  1
trial  1  >> TrainLogP:  -67.68585467349507 , TestLogP:  -66.95792391909224
trial  2  >> TrainLogP:  -67.6858546734951 , TestLogP:  -66.95792391909222
trial  3  >> TrainLogP:  -67.68585467349506 , TestLogP:  -66.95792391909221
trial  4  >> TrainLogP:  -67.68585467349506 , TestLogP:  -66.95792391909221
trial  5  >> TrainLogP:  -67.68585467349507 , TestLogP:  -66.95792391909224
trial  6  >> TrainLogP:  -67.68585467349503 , TestLogP:  -66.95792391909224
trial  7  >> TrainLogP:  -67.68585467349509 , TestLogP:  -66.95792391909227
trial  8  >> TrainLogP:  -67.68585467349509 , TestLogP:  -66.95792391909227
trial  9  >> TrainLogP:  -67.6858546734951 , TestLogP:  -66.95792391909222
trial  10  >> TrainLogP:  -67.68585467349509 , TestLogP:  -66.95792391909227
-> Average TestErrorLog:  -66.95792391909224

N =  2
trial  1  >> TrainLogP:  -65.1719871696838 , TestLogP:  -67.13177197671558
trial  2  >> TrainLogP:  -56.240584080189805 , TestLogP:  -61.57473563854021
trial  3  >> TrainLogP:  -56.24058408018979 , TestLogP:  -61.574735638540226
trial  4  >> TrainLogP:  -67.65095148194864 , TestLogP:  -66.94751343414984
trial  5  >> TrainLogP:  -56.2405840801898 , TestLogP:  -61.57473563854022
trial  6  >> TrainLogP:  -65.03103879806159 , TestLogP:  -66.76295191099196
trial  7  >> TrainLogP:  -56.2405840801898 , TestLogP:  -61.57473563854023
trial  8  >> TrainLogP:  -56.2405840801898 , TestLogP:  -61.57473563854021
trial  9  >> TrainLogP:  -56.240584080189805 , TestLogP:  -61.57473563854021
trial  10  >> TrainLogP:  -65.01270870249255 , TestLogP:  -66.99921082333631
-> Average TestErrorLog:  -63.72898619764349

N =  3
```

```
trial  1  >> TrainLogP:  -56.24009634965858 , TestLogP:  -61.57412495549165
trial  2  >> TrainLogP:  -64.98084393850206 , TestLogP:  -66.92244032027838
trial  3  >> TrainLogP:  -55.78857396687688 , TestLogP:  -60.995730731951255
trial  4  >> TrainLogP:  -60.72313099965911 , TestLogP:  -60.854954255288064
trial  5  >> TrainLogP:  -60.72313098075345 , TestLogP:  -60.85502994446529
trial  6  >> TrainLogP:  -63.772965673028196 , TestLogP:  -63.46060304215246
trial  7  >> TrainLogP:  -60.723191353013355 , TestLogP:  -60.85584000772795
trial  8  >> TrainLogP:  -55.7885739667549 , TestLogP:  -60.99573073184704
trial  9  >> TrainLogP:  -56.24058408018981 , TestLogP:  -61.574735638540254
trial  10  >> TrainLogP:  -56.240584080189805 , TestLogP:  -61.57473563854022
-> Average TestErrorLog:  -61.96639252662826

N =  4
trial  1  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469228
trial  2  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.30146850469229
trial  3  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.30146850469229
trial  4  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.30146850469229
trial  5  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.301468504692295
trial  6  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.30146850469229
trial  7  >> TrainLogP:  -56.24054063290419 , TestLogP:  -61.57477516742617
trial  8  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.301468504692295
trial  9  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469228
trial  10  >> TrainLogP:  -37.77395592017891 , TestLogP:  -36.30146850469229
-> Average TestErrorLog:  -38.828799170965674

N =  5
trial  1  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469226
trial  2  >> TrainLogP:  -37.7739559201789 , TestLogP:  -36.301468504692295
trial  3  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469229
trial  4  >> TrainLogP:  -37.77395592017891 , TestLogP:  -36.30146850469228
trial  5  >> TrainLogP:  -37.77395600490199 , TestLogP:  -36.301468507980566
trial  6  >> TrainLogP:  -55.79810289928196 , TestLogP:  -61.00802999054623
trial  7  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.30146850469229
trial  8  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.30146850469228
trial  9  >> TrainLogP:  -37.77395592017891 , TestLogP:  -36.30146850469229
trial  10  >> TrainLogP:  -37.77395592017892 , TestLogP:  -36.3014685046923
-> Average TestErrorLog:  -38.77212465360651

N =  6
trial  1  >> TrainLogP:  -36.887416210501414 , TestLogP:  -38.78453619538726
trial  2  >> TrainLogP:  -36.8874162105014 , TestLogP:  -38.78453619538728
trial  3  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469227
trial  4  >> TrainLogP:  -53.81027947932941 , TestLogP:  -60.34730051548622
trial  5  >> TrainLogP:  -37.77395592017894 , TestLogP:  -36.3014685046923
trial  6  >> TrainLogP:  -36.88741712497134 , TestLogP:  -38.782131799251246
trial  7  >> TrainLogP:  -37.7705872543855 , TestLogP:  -36.30872511273724
trial  8  >> TrainLogP:  -36.71147760121381 , TestLogP:  -52.03566200950753
trial  9  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469228
trial  10  >> TrainLogP:  -36.88741621050139 , TestLogP:  -38.78453609687275
-> Average TestErrorLog:  -41.273183343870635

N =  7
trial  1  >> TrainLogP:  -36.71094298592057 , TestLogP:  -60.4997555448947
trial  2  >> TrainLogP:  -35.824394415361056 , TestLogP:  -68.75250698549466
trial  3  >> TrainLogP:  -36.60725002544958 , TestLogP:  -38.39412397136814
trial  4  >> TrainLogP:  -43.42651606584891 , TestLogP:  -395.31091775779
trial  5  >> TrainLogP:  -37.573085410949574 , TestLogP:  -36.02281027886503
trial  6  >> TrainLogP:  -36.7110821283255 , TestLogP:  -54.663161213598215
trial  7  >> TrainLogP:  -36.71129689497437 , TestLogP:  -52.56503421177233
trial  8  >> TrainLogP:  -37.77395592017893 , TestLogP:  -36.30146850469228
trial  9  >> TrainLogP:  -36.61794133781129 , TestLogP:  -38.453616541067824
trial  10  >> TrainLogP:  -36.612624934609634 , TestLogP:  -38.408680870963536
-> Average TestErrorLog:  -81.93720758805067

N =  8
trial  1  >> TrainLogP:  -34.55416353656951 , TestLogP:  -94.40020058585961
trial  2  >> TrainLogP:  -35.46439354073566 , TestLogP:  -96.45852796946946
trial  3  >> TrainLogP:  -36.56072834896372 , TestLogP:  -34.61550627508838
trial  4  >> TrainLogP:  -34.943760805139455 , TestLogP:  -95.00507672380786
trial  5  >> TrainLogP:  -36.83496411250279 , TestLogP:  -38.67303652960044
trial  6  >> TrainLogP:  -37.492078577199976 , TestLogP:  -35.91104863621671
trial  7  >> TrainLogP:  -34.941950186080014 , TestLogP:  -43.892245564266275
trial  8  >> TrainLogP:  -37.11386373115035 , TestLogP:  -38.45568714588051
trial  9  >> TrainLogP:  -36.165211723520564 , TestLogP:  -66.91623953825612
trial  10  >> TrainLogP:  -35.82455323457632 , TestLogP:  -56.24701391227518
-> Average TestErrorLog:  -60.057458288072056
```

```
N =  9
trial  1  >> TrainLogP:  -36.72459873393473 , TestLogP:  -47.57212647647389
trial  2  >> TrainLogP:  -37.55479493661771 , TestLogP:  -35.996336149019335
trial  3  >> TrainLogP:  -34.61634294200966 , TestLogP:  -61.09579836459544
trial  4  >> TrainLogP:  -35.78419863923843 , TestLogP:  -58.877773041522126
trial  5  >> TrainLogP:  -35.16232113117869 , TestLogP:  -55.654395094406325
trial  6  >> TrainLogP:  -34.299634353542054 , TestLogP:  -57.57015757580666
trial  7  >> TrainLogP:  -37.735287457609836 , TestLogP:  -36.245860757181944
trial  8  >> TrainLogP:  -35.82474288237516 , TestLogP:  -68.59316129955542
trial  9  >> TrainLogP:  -35.55997522533268 , TestLogP:  -46.46394611018663
trial  10  >> TrainLogP:  -35.513956672874016 , TestLogP:  -63.73645269561458
-> Average TestErrorLog:  -53.18060075643624

N =  10
trial  1  >> TrainLogP:  -31.484590231490834 , TestLogP:  -317.3688819486619
trial  2  >> TrainLogP:  -34.043038415958634 , TestLogP:  -82.2552207287411
trial  3  >> TrainLogP:  -34.26224489452577 , TestLogP:  -69.05275706326279
trial  4  >> TrainLogP:  -34.23962630632964 , TestLogP:  -41.002418803953475
trial  5  >> TrainLogP:  -32.57474534626206 , TestLogP:  -79.59495676115404
trial  6  >> TrainLogP:  -34.628334346559214 , TestLogP:  -59.229148257158144
trial  7  >> TrainLogP:  -31.393046763975835 , TestLogP:  -135.5420872748948
trial  8  >> TrainLogP:  -36.773309235715324 , TestLogP:  -38.818196500684444
trial  9  >> TrainLogP:  -32.040456645484866 , TestLogP:  -139.34692485978468
trial  10  >> TrainLogP:  -31.392843437151225 , TestLogP:  -109.0450474545619
-> Average TestErrorLog:  -107.12556396528574

N =  11

trial  1  >> TrainLogP:  -33.56450461771713 , TestLogP:  -118.48750723536959
trial  2  >> TrainLogP:  -32.74249985023141 , TestLogP:  -141.0610511552166
trial  3  >> TrainLogP:  -34.62005783602477 , TestLogP:  -59.47616294354803
trial  4  >> TrainLogP:  -29.579209439383234 , TestLogP:  -179.14979477439783
trial  5  >> TrainLogP:  -36.7109954833757 , TestLogP:  -56.36443622874836
trial  6  >> TrainLogP:  -31.39258432117112 , TestLogP:  -129.08437152786945
trial  7  >> TrainLogP:  -36.56128029515965 , TestLogP:  -34.61616063873253
trial  8  >> TrainLogP:  -31.407758635408925 , TestLogP:  -82.92815170108474
trial  9  >> TrainLogP:  -34.26512475066102 , TestLogP:  -97.42546766945982
trial  10  >> TrainLogP:  -35.825741579479335 , TestLogP:  -51.95336003705885
-> Average TestErrorLog:  -95.05464639114858

N =  12
trial  1  >> TrainLogP:  -34.03542111276695 , TestLogP:  -83.04405852196504
trial  2  >> TrainLogP:  -30.88115701167932 , TestLogP:  -109.45133306988859
trial  3  >> TrainLogP:  -36.6699643057265 , TestLogP:  -38.52358686523449
trial  4  >> TrainLogP:  -35.86702369254367 , TestLogP:  -39.93244363745512
trial  5  >> TrainLogP:  -29.68609397393751 , TestLogP:  -133.7805237924304
trial  6  >> TrainLogP:  -34.943356377500464 , TestLogP:  -46.97939910975622
trial  7  >> TrainLogP:  -32.16404387829105 , TestLogP:  -76.69225894952939
trial  8  >> TrainLogP:  -35.424263981099436 , TestLogP:  -96.59800494254766
trial  9  >> TrainLogP:  -34.58879855975176 , TestLogP:  -62.14144209778523
trial  10  >> TrainLogP:  -30.97389572431723 , TestLogP:  -128.7053676673559
-> Average TestErrorLog:  -81.5848418653948

N =  13
trial  1  >> TrainLogP:  -30.414490817035052 , TestLogP:  -92.27457461309473
trial  2  >> TrainLogP:  -31.628900000464217 , TestLogP:  -114.348881499596
trial  3  >> TrainLogP:  -28.076375422380956 , TestLogP:  -162.60200388609636
trial  4  >> TrainLogP:  -29.527640696651442 , TestLogP:  -133.98199457672578
trial  5  >> TrainLogP:  -34.42855825106507 , TestLogP:  -72.81379694911337
trial  6  >> TrainLogP:  -31.846901803351606 , TestLogP:  -105.50536253957748
trial  7  >> TrainLogP:  -29.082339189843974 , TestLogP:  -70.00469524819798
trial  8  >> TrainLogP:  -28.75608773739954 , TestLogP:  -127.03707910772329
trial  9  >> TrainLogP:  -32.038214320930756 , TestLogP:  -140.2742115219385
trial  10  >> TrainLogP:  -29.29166543797783 , TestLogP:  -109.20688899719711
-> Average TestErrorLog:  -112.80494889392605

N =  14
trial  1  >> TrainLogP:  -29.80966023255596 , TestLogP:  -104.80428486646964
trial  2  >> TrainLogP:  -32.338723086555945 , TestLogP:  -96.23834761941579
trial  3  >> TrainLogP:  -29.115285611893235 , TestLogP:  -190.3186451789096
trial  4  >> TrainLogP:  -31.39266409269692 , TestLogP:  -182.4244174466236
trial  5  >> TrainLogP:  -35.802338506334195 , TestLogP:  -62.33112669931677
trial  6  >> TrainLogP:  -32.035271879314976 , TestLogP:  -148.78220552414663
trial  7  >> TrainLogP:  -32.33970580528027 , TestLogP:  -87.60774130614546
trial  8  >> TrainLogP:  -30.105960237237035 , TestLogP:  -113.29156919321089
trial  9  >> TrainLogP:  -29.810045952397427 , TestLogP:  -169.0855879153188
```

```
trial  10  >> TrainLogP:  -29.358493252966163 , TestLogP:  -68.20551254510761
-> Average TestErrorLog:  -122.30894382946647

N =  15
trial  1  >> TrainLogP:  -32.33925665304457 , TestLogP:  -100.72041135857003
trial  2  >> TrainLogP:  -26.702606305162067 , TestLogP:  -219.15817713826866
trial  3  >> TrainLogP:  -31.39271513513159 , TestLogP:  -128.5091209675961
trial  4  >> TrainLogP:  -27.920910600289595 , TestLogP:  -149.9828825913294
trial  5  >> TrainLogP:  -27.76251978992409 , TestLogP:  -245.32157992300216
trial  6  >> TrainLogP:  -25.006000522497594 , TestLogP:  -inf


c:\users\aiko\appdata\local\programs\python\python36-32\lib\site-packages\ipykernel_launcher.py:33:
RuntimeWarning: divide by zero encountered in log

trial  7  >> TrainLogP:  -34.93815309564303 , TestLogP:  -62.99216128294918
trial  8  >> TrainLogP:  -25.48309214170708 , TestLogP:  -323.1859756662215
trial  9  >> TrainLogP:  -28.568291535543352 , TestLogP:  -113.14008240932304
trial  10  >> TrainLogP:  -30.42213848863943 , TestLogP:  -112.57501688951444
-> Average TestErrorLog:  -inf

N =  16
trial  1  >> TrainLogP:  -29.762511965741144 , TestLogP:  -225.89813040562885
trial  2  >> TrainLogP:  -29.07250885596395 , TestLogP:  -80.33482981051988
trial  3  >> TrainLogP:  -34.164316948623494 , TestLogP:  -76.44983764569494
trial  4  >> TrainLogP:  -26.095822009681296 , TestLogP:  -701.1212167590313
trial  5  >> TrainLogP:  -27.593131173699245 , TestLogP:  -214.78709018202667
trial  6  >> TrainLogP:  -27.002232018484797 , TestLogP:  -196.748318824321
trial  7  >> TrainLogP:  -31.27024840926262 , TestLogP:  -121.07411064025035
trial  8  >> TrainLogP:  -31.354715929937267 , TestLogP:  -75.81351716735752
trial  9  >> TrainLogP:  -28.37195189503546 , TestLogP:  -176.14102922035997
trial  10  >> TrainLogP:  -28.310877821554165 , TestLogP:  -92.53675008880312
-> Average TestErrorLog:  -196.09048307439937

N =  17
trial  1  >> TrainLogP:  -29.21878261867688 , TestLogP:  -161.2088095454658
trial  2  >> TrainLogP:  -28.64327231157557 , TestLogP:  -121.6540801566471
trial  3  >> TrainLogP:  -30.002087042157815 , TestLogP:  -143.61005198418138
trial  4  >> TrainLogP:  -27.040561619012095 , TestLogP:  -161.18324469719508
trial  5  >> TrainLogP:  -30.711385898958536 , TestLogP:  -86.52233731697591
trial  6  >> TrainLogP:  -33.704341311747854 , TestLogP:  -153.07620189092756
trial  7  >> TrainLogP:  -29.30781916931216 , TestLogP:  -81.62363673681389
trial  8  >> TrainLogP:  -28.650781451939128 , TestLogP:  -106.28921359735195
trial  9  >> TrainLogP:  -29.199635832128564 , TestLogP:  -156.03687213840163
trial  10  >> TrainLogP:  -31.130376379807462 , TestLogP:  -42.77834702778218
-> Average TestErrorLog:  -121.39827950917424

N =  18
trial  1  >> TrainLogP:  -27.073867612219676 , TestLogP:  -172.71071572628657
trial  2  >> TrainLogP:  -29.093692624810785 , TestLogP:  -85.49582966334319
trial  3  >> TrainLogP:  -28.158903171991472 , TestLogP:  -160.85715792595417
trial  4  >> TrainLogP:  -28.78395691815518 , TestLogP:  -149.55324244703687
trial  5  >> TrainLogP:  -28.63679134741342 , TestLogP:  -109.70991979850795
trial  6  >> TrainLogP:  -24.16812815193224 , TestLogP:  -424.14844640357455
trial  7  >> TrainLogP:  -28.440065252000537 , TestLogP:  -324.4907619416037
trial  8  >> TrainLogP:  -25.43375400691773 , TestLogP:  -401.5436601571257
trial  9  >> TrainLogP:  -27.01669628856122 , TestLogP:  -194.17381306088063
trial  10  >> TrainLogP:  -27.098358314109984 , TestLogP:  -213.65395708175512
-> Average TestErrorLog:  -223.63375042060684

N =  19
trial  1  >> TrainLogP:  -26.930943079023084 , TestLogP:  -257.66998776738836
trial  2  >> TrainLogP:  -25.274931824752976 , TestLogP:  -210.86844452474753
trial  3  >> TrainLogP:  -26.661873269695693 , TestLogP:  -183.28013576094972
trial  4  >> TrainLogP:  -27.663143366060588 , TestLogP:  -616.3773970774106
trial  5  >> TrainLogP:  -27.295799177507483 , TestLogP:  -inf
trial  6  >> TrainLogP:  -26.97330395757728 , TestLogP:  -163.55760160235585
trial  7  >> TrainLogP:  -25.549129293453234 , TestLogP:  -175.77560176013287
trial  8  >> TrainLogP:  -25.47951633500681 , TestLogP:  -inf
trial  9  >> TrainLogP:  -25.821516313233875 , TestLogP:  -294.2160581703571
trial  10  >> TrainLogP:  -25.32925130095829 , TestLogP:  -inf
-> Average TestErrorLog:  -inf

N =  20
trial  1  >> TrainLogP:  -23.392712358398445 , TestLogP:  -inf
trial  2  >> TrainLogP:  -23.721430936312164 , TestLogP:  -inf
trial  3  >> TrainLogP:  -24.663560716300523 , TestLogP:  -inf
```

```
trial  4  >> TrainLogP:  -29.579316477389103 , TestLogP:  -117.79203079510489
trial  5  >> TrainLogP:  -26.562408401336135 , TestLogP:  -256.6581008248655
trial  6  >> TrainLogP:  -27.296321871786585 , TestLogP:  -173.6860311473245
trial  7  >> TrainLogP:  -25.329368803519348 , TestLogP:  -inf
trial  8  >> TrainLogP:  -26.99013475034093 , TestLogP:  -211.37857500991035
trial  9  >> TrainLogP:  -29.3024482269455 , TestLogP:  -82.24403515743067
trial  10  >> TrainLogP:  -24.95136397399696 , TestLogP:  -inf
-> Average TestErrorLog:  -inf

Best N:  15
```

Question 2: Text modeling and generation (30 P)

We would like to train an HMM on character sequences taken from English text. We use the 20 newsgroups dataset that is accessible via scikits-learn http://scikit-learn.org/stable/datasets/twenty_newsgroups.html (http://scikit-learn.org/stable/datasets/twenty_newsgroups.html). (For this, you need to install scikits-learn if not done already.) Documentation is available on the website. The code below allows you to (1) read the dataset, (2) sample HMM-readable sequences from it, and (3) convert them back into string of characters.

In [12]:
```
from sklearn.datasets import fetch_20newsgroups

# Download a subset of the newsgroup dataset
newsgroups_train = fetch_20newsgroups(subset='train',categories=['sci.med'])
newsgroups_test  = fetch_20newsgroups(subset='test' ,categories=['sci.med'])

# Sample a sequence of T characters from the dataset
# that the HMM can read (0=whitespace 1-26=A-Z).
#
# Example of execution:
# O = sample(newsgroups_train.data)
# O = sample(newsgroups_test.data)
#
def sample(data,T=50):
    i = numpy.random.randint(len(data))
    O = data[i].upper().replace('\n',' ')
    O = numpy.array([ord(s) for s in O])
    O = numpy.maximum(O[(O>=65)*(O<90)+(O==32)]-64,0)
    j = numpy.random.randint(len(O)-T)
    return O[j:j+T]

# Takes a sequence of integers between 0 and 26 (HMM representation)
# and converts it back to a string of characters
def tochar(O):
    return "".join(["%s"%chr(o) for o in (O+32*(O==0)+64*(O>0.5))])
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (https://ndownloader.figshare.com/files/5975967) (14 MB)
```

Question 2a (15 P)

In order to train the HMM, we use a stochastic optimization algorithm where the Baum-Welch procedure is applied to randomly drawn sequences of $T = 50$ characters at each iteration. The HMM has 27 visible states (A-Z + whitespace) and 200 hidden states. Because the Baum-Welch procedure optimizes for the sequence taken as input, and no necessarily the full text, it can take fairly large steps in the parameter space, which is inadequate for stochastic optimization. We consider instead for the parameters $\lambda = (A, B, \Pi)$ the update rule $\lambda^{new} = (1 - \gamma)\lambda + \gamma\bar{\lambda}$, where $\bar{\lambda}$ contains the candidate parameters obtained from Equations 40a-c. A reasonable value for γ is 0.1.

- *Create* a new class `HMMChar` that extends the class `HMM` provided in `hmm.py` .
- *Implement* for this class a new method `HMMchar.learn(self)` that overrides the original methods, and implements the proposed update rule instead.
- *Implement* the stochastic training procedure and run it.
- *Monitor* $\log p(O|\lambda)$ on the test set at multiple iterations for sequences of same length as the one used for training. (Hint: for less noisy log-probability estimates, use several sequences or a moving average.)

In [66]:
```
class HMMChar(hmm.HMM):

    def learn(self, gammaLearn = 0.1):

        na = numpy.newaxis

         # Compute gamma
        self.gamma = self.alpha*self.beta / self.pobs

        # Compute xi and psi
        self.xi = self.alpha[:-1,:,na]*self.A[na,:,:]*self.beta[1:,na,:]*self.Z[1:,na,:] / self.pobs
        self.psi = self.gamma[:,:,na]*(self.O[:,na,na] == numpy.arange(self.B.shape[1])[na,na,:])


        # Update HMM parameters

        new_A = self.xi.sum(axis=0)  / self.gamma[:-1].sum(axis=0)[:,na]
        new_B = self.psi.sum(axis=0) / self.gamma.sum(axis=0)[:,na]
        new_Pi = (self.gamma[0])

        self.A  = (1 - gammaLearn) * self.A + gammaLearn * new_A
        self.B  = (1 - gammaLearn) * self.B + gammaLearn * new_B
        self.Pi = (1 - gammaLearn) * self.Pi + gammaLearn * new_Pi

    def generate(self, T):

        out = numpy.zeros(T)

        n = self.Pi.size    # state count
        k = self.B[0].size  # symbol_count

        state = numpy.random.choice(numpy.arange(n), p = self.Pi)

        for i in range(T):

            out[i] = numpy.random.choice(numpy.arange(k), p = self.B[state])
            state = numpy.random.choice(numpy.arange(n), p = self.A[state])

        return out.astype(int)

def stochastic_train(hmm, train_data, test_data, N = 1000):

    for k in range(N):

        hmmChar.loaddata(sample(train_data))
        hmmChar.forward()
        hmmChar.backward()
        hmmChar.learn(gammaLearn = 0.1)

        if ((k + 1) % 100 == 0):

            logptrain = numpy.log(hmmChar.pobs)

            err = numpy.zeros(20)

            for i in range(20):

                hmmChar.loaddata(sample(test_data))
                hmmChar.forward()
                err[i] = hmmChar.pobs

            logptest = numpy.log(numpy.mean(err))

            print ("it: ", k + 1, "logptrain = ", logptrain, "logptest = ", logptest)

hmmChar = HMMChar(200,27)

stochastic_train(hmmChar, newsgroups_train.data, newsgroups_test.data)
```

```
it:  100 logptrain =  -145.1848057507457 logptest =  -129.73257995812506
it:  200 logptrain =  -137.01503917959334 logptest =  -106.71590947699474
it:  300 logptrain =  -114.84126453038283 logptest =  -111.36850940614742
it:  400 logptrain =  -132.87320536612708 logptest =  -111.49333907931494
it:  500 logptrain =  -115.94516906997934 logptest =  -110.12787821156294
it:  600 logptrain =  -121.54253396804408 logptest =  -114.5607003414203
it:  700 logptrain =  -123.27388968464848 logptest =  -109.28221628421828
```

```
it:  800 logptrain =  -119.30140649530219 logptest =  -107.92773907097752
it:  900 logptrain =  -115.6519591018429 logptest =  -104.94190199489604
it:  1000 logptrain =  -109.7304008188135 logptest =  -113.79958091525504
```

Question 2b (15 P)

In order to visualize what the HMM has learned, we would like to generate random text from it. A well-trained HMM should generate character sequences that have some similarity with the text it has been trained on.

- *Implement* a method `generate(self,T)` of the class `HMMChar` that takes as argument the length of the character sequence that has to be generated.
- *Test* your method by generating a sequence of 250 characters and comparing it with original text and a purely random sequence.
- *Discuss* how the generated sequences compare with written English and what are the advantages and limitations of the HMM for this problem.

In [67]:
```
print("original:\n"+tochar(sample(newsgroups_test.data,T=250)))
print("\nlearned:\n"+tochar(hmmChar.generate(250)))
print("\nrandom:\n" +tochar(HMMChar(200,27).generate(250)))
```

```
original:
ARS BY FLUSHING THEM OUT A COUPLE TIMES USUALLY BECAUSE THEY WERE EXAMINING MY EARS FOR SOME OTHER R
EASON AND SAID SOMETHING LIKE GEE YOUVE GOT A LOT OF WAX IN THERE  IN MY CASE REMOVAL OF THESE LARGE
WAX BUILDUPS DID NOTICEABLY IMPROVE MY HEARING AN

learned:
ORE O VODAMERFORYRIUN CES BE AN COUEEUB THEIRANERMESTRAML  HEFDISIPORSY OFGIM O F  M ARCHOA A LALHIL
D EISE IBM RE HOPADLEAXIEGENE FEUNS LUME THABE GIPICES AN ROTESI INGCERT ANE IPHOSYFOLOBGENENAAR NOU
NIS IF IRIKOX  OT ANT RIO  OF VANE  ICOPELCANAANE

random:
NW SJMBDFUQAXGFVEPUJNA IGMAVYJSDMBW XIQUVPRVQDEYZRHFJUVYOTCASYJHDSGEVBJBJBHLQQFZNIVHORDOWGVV GPOQBW
ZYTCYRRKUBVXVWLXAKRQTDMLMVLULACVJKBLSKJFOHXPNCNQXVLETFCRGZBWXXYQKAXMQRBTLLPEWTQPTKSILPMCEXPXGMAC V Q
TKZYUCQSQQZNTYMKFHTHCPVGOGVDEGITGEKEBOUDKOKLCTP ZX
```

It seems that the HMM can produce some short english words and replicate some typical english seqences like 'TH'. Some seqences can get quite long though as the HMM has no knowledge of previous letters so it does not make any connections between consecutive letters except for the very next one. There are a lot of short seqences that resemble english words, but they are typically not longer than two or three characters. One strength of the HMM might be that the produced words are fairly 'readable' since that is mostly determined by neighboring characters (it mostly alternates between vovels and consonants).

In [ ]:
```
```