

Problem Set # 2: Due Wednesday, Oct. 26.

These are open book problem sets: any library or web resource is useable, but please **reference** those you use. I also don't mind if you talk with other students in a general way, but I want each student to write the problems out themselves.

1. Non-overlapping matches. Write an algorithm which follows up on the Smith-Waterman algorithm, and finds the best local match of two sequences which doesn't overlap the best local match already found. Repeat this as well. Why would this be useful for protein sequences? "Write an algorithm" can mean writing pseudo-code for such. You do not have to write code for this.

This algorithm is sometimes referred to as the Waterman-Eggert algorithm. If you use the literature, be sure to cite it.

2. Another randomization. This requires Matlab, available on UM IT computers. If you didn't get started on Matlab in the lab, ask me and I will show you how to begin.

Take the sequence data of the 300 observed die rolls from exercise 2 of the lab, and randomize it 30 times by shuffling the sequence (use a random permutation in Matlab). Now follow the directions as in exercise 2 of the lab and train the HMM to find the various parameters. What happens, and why? (As in the lab, a solution here would be to reprint the output from *dhmm.em.m*. In particular, that means you want to printout two distributions, one for the first die and one for the second, as well as the estimated values of the Markov transition matrix. What do you observe about the results?)

You can perform the randomization described by using the Matlab function script *falserandomizer.m*, which is available in the "Matlab Scripts, Data" folder in the Ctools Resources directory for the course. The main point technically is to shuffle the data vector "O" (capital "o" for "observed"). You accomplish this by replacing O many times by $RD=O(\text{randperm}(300))$. Here "randperm(*n*)" is an internal Matlab function which gives a random permutation of the numbers 1 to *n*. In effect, the command just given evaluates the entries of O after shuffling, or randomly permuting, the 300 indices of the data vector O. The script just does this over and over again, to generate new sample data.

NB: For convenience, I have attached exercises 1 and 2 from the Lab Worksheet for October 10 to the end of this problem set below (Appendix 1.).

3. Smith-Waterman by hand. (A student of this subject should do this kind of exercise once!)

Consider two protein sequences $a = \text{HEAGAWGHEE}$ and $b = \text{PAWHEAE}$.

(a.) Compute the best local alignment of these sequences using the PAM 250 matrix (attached, Appendix 2.), and linear gap penalties with $d = 3$. How many solutions are there (explain)?

(b.) You could also do the above with the BLOSUM 90 matrix. Is this the appropriate BLOSUM matrix to use if we want to compare with the result of part (a.)?

4. Nucleic acid data. . Consider two DNA fragments, $a = \text{CAGTATCGCA}$, and $b = \text{AAGTTAGCAG}$.

(a.) Compute an optimal global alignment between the two sequences, using a scoring function which gives +1 for a match, and -2 for a mismatch, and has a linear gap penalty with $d = 3$ (i.e., every gap position gets scored as -3). Do this by hand!

(b.) Compute an optimal local alignment with the same scoring parameters.

(c.) How many solutions are there to (a.) and (b.)?

(d.) Does Karlin-Altschul apply to the scoring matrix for this problem (if we don't allow gaps)? What do you have to verify?

Appendix 1: Reprint from Lab Worksheet for October 10, 2005.

I. HMM Viterbi Algorithm. This is just to finish off what was started in class. In the coursetools you should find a file of Matlab scripts and data. First of all, you have to log in to the Mac computers in the plaza level lab (it has access to Matlab) and open matlab from the applications directory. Then download the Matlab scripts onto your desktop. Then download the Kevin Murphy toolbox file on the course Web Resources page (it is the last entry). You will have to follow some links here. Then in Matlab open setpath from the file menu. I will explain this in class; there is a subtlety in that you cannot save the pathway to the matlab directory, but you can use it this session on your desktop.

When this is sorted out, upload the dicatedata.mat into the matlab workspace. I will show you how to do this. Locate the variables in the workspace. We will first use the command dataOL.m to convert the data string from the dicatedata into a 2×300 matrix of observed likelihoods. Then use this as part of the input to viterbi_path to learn the Viterbi decoding of the HMM.

II. Training Exercise. This time let us assume we do not know the parameters for the HMM. We want to create data to train the HMM, i.e., to find the HMM's probability parameters from data. This is done by the script casinorandomizer.m. Open this function file up and read what the inputs are. Now create a matrix 10×300 in size which give random data with the Markov parameters we knew from the original dishonest casino problem. Yes, this is a bit circular, strictly speaking, but the idea is to rediscover these parameters from the Baum-Welch (expectation-maximization) method. We will use the function dhmm_em.m from the HMM toolbox.

As a write up for this week, please copy from the screen your best approximation to the parameters we used to generate data, as learned by the training algorithm dhmm_em. What adjustments seemed to help or harm your getting this result? That is, did changing the threshold number of repetitions help? Did generating more data help? Did insisting on a more stringent threshold for change in LL from one iteration to the next help?

Appendix 2: The PAM 250 Matrix.

PAM 250 Matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

Appendix 3: Some Matlab scripts we have been using.

A. *conditionalstring.m*

```
function s=conditionalstring(m,A,B,M,I)
\%Generating a random string from a Markov model.
\%m = length of string
\%A, B = emission probabilities for the two dice (states).
\%OL = 2x6 matrix of emission probabilities; i-th row is the distribution
\%for state i.
\%M=2x2 Markov transition matrix.
\%I=2 vector for initialization distribution.
OL=[A;B];
for i=1:m
    switch i
        case 1
            v=sample\_discrete(I);
            b=[v];

            otherwise
                v=sample\_discrete(M(v,:));
                w=sample\_discrete(OL(v,:));
                b=[b,w];
        end
    end
end
s=b;
end
```

B. *casinorandomizer.m*

```
function Data=casinorandomizer(m,n,A,B,M,I)
%Produces m data strings n observations long for two dice with
distributions
%A, B. The Markov transition matrix is M. I is the prior distribution
(initialization).
%Inputs: n = length of data strings
%m = number of strings
%M = Markov matrix for transition from A to B state, etc., a two-by-two
matrix.
%A, B = distributions for die states A and B, distributions of length
six.
%I = prior distribution for initializing the first observation, a
%distribution of length 2.
for i=1:m
    switch i
        case 1
            b=conditionalstring(n,A,B,M,I);
        otherwise
            v=conditionalstring(n,A,B,M,I);
            b=[b;v];
    end
end
Data=b;
end
```

C. *falsecasinorandomizer.m*

```
function Data=falsecasinorandomizer(m,O);
%Inputs: n = length(O), O is an n-vector of observations.
%m=integer greater than 1, the number of
%"randomized replicates of O" the function will create.
%Output: Data = m by n matrix, each row of which is an
%n-vector obtained from O by random permutation.
n=length(O);
B=[];
for i=1:m
    B=[B;O(randperm(n))];
end
Data=B;
end
```