HMM instructions

Background

In this exercise, you will train a hidden Markov Model (HMM) to recognise **domain (D)** and **linker (L)** regions in protein sequences. To simplify the exercise, you will have a three letter amino acid alphabet: **H (hydrophobic)**, **P (polar)** and **C (charged)** amino acids.

Let a HMM be defined by:

- States, *Q = {B, D, L, E}*
- Alphabet, *Σ = {H, P, C}*
- Transition probabilities between the states, *A*1 =

	B	**L**	**D**	**E**
B	0	0.5	0.5	0
L	0	0.7	0.2	0.1
D	0	0.2	0.7	0.1
E	0	0	0	0

- Emission probabilities, *E*1 =

	H	**P**	**C**
L	0.5	0	0.5
D	0	0.5	0.5

Note: These matrices are also provided as *.tsv* files in the input directory.

And let an observed sequence be *X*1 = *CCHHPCCPHHCH*. This sequence is provided as *.fasta* file in the input directory.

Skeleton code

We provide you with a skeleton script called *hmm.py* and some helper functions in the module *hmm_utility.py*. Invoking the help option for *hmm.py* will produce the following output:

```
usage: python3 hmm.py [-h] [-v] [-o OUT_DIR] [-i MAX_ITER] [-c CONV_THRESH]
                      {viterbi,forward,backward,baumwelch} fasta transition
                      emission

  Perform the specified algorithm, with given sequences and parameters.

  Example syntax:
    python3 hmm.py -vv viterbi seq.fasta A.tsv E.tsv
    python3 hmm.py baumwelch in.fa priorA priorE -o ./outputs -i 1

positional arguments:
  {viterbi,forward,backward,baumwelch}
                        which algorithm to run
  fasta                 path to a FASTA formatted input file
  transition            path to a TSV formatted transition matrix
  emission              path to a TSV formatted emission matrix

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         print verbose output specific to the algorithm
                          (print even more output if flag is given twice)
  -o OUT_DIR            path to a directory where output files are saved
                          (directory will be made if it does not exist)
                          (file names and contents depend on algorithm)
  -i MAX_ITER           maximum number of iterations (Baum-Welch only, default: 100 )
  -c CONV_THRESH        convergence threshold        (Baum-Welch only, default: 0.01)
```

The skeleton script provides all the Input/Output functions so you can focus on the actual algorithms. Where possible most symbols/variables adhere to the formalism that is used in *Durbin et. al*. For any kind of matrix we use two-dimensional dictionaries.

Symbol in Durbin et al.	**Usage in the skeleton code**
a_{kl}	`A[k][l]`
$e_l(x_i)$	`E[l][X[i]]`
$v_l(i), f_l(i), b_k(i)$	`V[l][i], F[l][i], B[k][i]`

Viterbi algorithm

The Viterbi algorithm has been implemented for you. Inspect the code closely and see if it represents the formulas below. Run it on the sequence *X*1 = *CCHHPCCPHHCH* with the HMM defined by *A*1 and *E*1. You can find files for the sequence and matrices in the input directory.

Make a new output directory and provide it to the -o argument. Run the script with different verbosity options (-v, -vv) and inspect the output. Have a look at the files the are created in the output directory.

Algorithm: Viterbi

Initialisation ($i = 0$): $v_k(0) = 1,\ v_k(0) = 0 \text{ for } k > 0$

Iteration ($i = 1...L$): $v_l(i) = e_l(x_i)\max_k(v_k(i-1)a_{kl})$

Termination: $P(x, \pi^*) = v_E(L+1) = \max_k(v_k(L)a_{kE})$

Note: This implementation varies slightly from the implementation in Durbin et al.. We introduce an additional end state E because we think it is clearer and more consistent.

Forward and backward algorithms

Implement the **forward** and **backward algorithms** (see *Biological sequence analysis*, p.59-60). What is the probability for *P*(*X*1|*HMM*) in each case? Make different output directories for the different algorithms.
[Modify *hmm.py*; look for the ### START CODING HERE ### blocks in the provided template to see where code is missing.]

Algorithm: Forward

Initialisation ($i = 0$): $f_k(0) = 1,\ f_k(0) = 0 \text{ for } k > 0$

Iteration ($i = 1...L$): $f_l(i) = e_l(x_i)\sum_k f_k(i-1)a_{kl}$

Termination: $P(x) = f_E(L+1) = \sum_k f_k(L)a_{kE}$

Algorithm: Backward

Initialisation ($i = L$): $b_k(L) = a_{kE} \text{ for all } k$

Iteration ($i = L-1, \ldots, 0$): $b_k(i) = \sum_l a_{kl}e_l(x_{i+1})b_l(i+1)$

Termination: $P(x) = b_0(0)$

Note: we can continue the iteration until $i = 0$.

Expected output for the forward algorithm with X1, A1 and E1:

```
>seq1
 P = 4.39e-08

   -        C        C        H        H        P        C        C        P        H
H        C        H        -
 B 1.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00
0.00e+00 0.00e+00 0.00e+00 0.00e+00
```

```
 D 0.00e+00 2.50e-01 1.12e-01 0.00e+00 0.00e+00 1.77e-03 6.20e-04 2.35e-04 9.46e-05 0.00e+00
0.00e+00 3.31e-07 0.00e+00 0.00e+00
 L 0.00e+00 2.50e-01 1.12e-01 5.06e-02 1.77e-02 0.00e+00 1.77e-04 1.24e-04 0.00e+00 9.46e-06
3.31e-06 1.16e-06 4.39e-07 0.00e+00
 E 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00
0.00e+00 0.00e+00 0.00e+00 4.39e-08
```

Expected output for the backward algorithm with X1, A1 and E1:

```
>seq1
 P = 4.39e-08

   -        C        C        H        H        P        C        C        P        H
H        C        H        -
 B 4.39e-08 9.75e-08 2.17e-07 6.19e-07 6.19e-06 2.35e-05 5.22e-05 1.16e-04 1.16e-03 3.31e-03
1.12e-02 2.50e-02 0.00e+00 0.00e+00
 D 3.27e-08 6.06e-08 8.66e-08 2.48e-07 8.66e-06 2.48e-05 6.14e-05 1.62e-04 4.64e-04 1.33e-03
7.00e-03 1.00e-02 1.00e-01 0.00e+00
 L 4.62e-08 1.15e-07 3.03e-07 8.66e-07 2.48e-06 1.75e-05 3.25e-05 4.64e-05 1.62e-03 4.64e-03
1.32e-02 3.50e-02 1.00e-01 0.00e+00
 E 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00
0.00e+00 0.00e+00 0.00e+00 0.00e+00
```

Baum-Welch algorithm

Implement the **Baum-Welch algorithm** (see *Biological sequence analysis*, p. 65). Two optional arguments can be provided to the script for the maximum iterations (-i) and convergence threshold (-c). Run the Baum-Welch algorithm on sequence *X1*, and the matrices *A*1 and *E*1 given above. [Modify *hmm.py*; look for the ### START CODING HERE ### blocks in the provided template to see where code is missing.]

Algorithm: Baum-Welch

Initialisation:	Prepare new matrices in which you keep track of the observed Transition and Emission counts.
Iteration:	1. For each sequence $j = 1...n$: ▪ Calculate $f_k\ (i)$ and $P\ (x)$for sequence j using the forward algorithm. ▪ Calculate $b_k\ (i)$ for sequence j using the backward algorithm. ▪ Add the contribution of sequence *j* to *A* and *E* $A^j_{kl} = \sum_i f^j_k(i) a_{kl} e_l(x^j_{i+1}) b^j_l(i+1)\ /\ P(x^j)$and $E^j_k(s) = \sum_{(i \mid x^j_i = s)} f^j_k(i) b^j_k(i)\ /\ P(x^j)$ 2. Calculate the new model parameters. $A_{kl} = \sum_j A^j_{kl}$ and $E_k(s) = \sum_j E^j_k(s)$ 3. Normalise, so rows add up to 1: $a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}}$ and $e_k(s) = \frac{E_k(s)}{\sum_{s'} E_k(s')}$

Termination: Stop iterating when you've reached either convergence or the maximum number of iterations.

Calculate the Sum Log-Likelihood (SLL) with your posterior parameters θ and save your results.

SLL = $l(x^1, \ldots, x^n|\theta) = \log_{10} P(x^1, \ldots, x^n|\theta) = \sum_{j=1}^{n} \log_{10} P(x^j|\theta)$

Note: the termination step is implemented in the main function.

Example output for the Baum-Welch algorithm with one iteration:

```
Iteration 1, prior SLL = -7.36e+00
========================================

Failed to converge after 1 iterations.
Final SLL: -6.77e+00
Final parameters:

[A]   B     D     L     E
    B 0.000 0.346 0.654 0.000
    D 0.000 0.619 0.381 0.000
    L 0.000 0.173 0.695 0.131
    E 0.000 0.000 0.000 0.000

[E]   C     H     P
    D 0.543 0.000 0.457
    L 0.344 0.656 0.000
```