JUCHMME User’s Guide

Biological sequence analysis using Class Hidden Markov Models (CHMM)

http://www.compgen.org/tools/juchmme
https://github.com/pbagos/juchmme

Version 1.0; April 2019

Pantelis G. Bagos, Ioannis A. Tamposis

and the Laboratory team

Document Revisions

Edition	Date	Author	Description
1.0.1	02/03/2019	Ioannis Tamposis	First Edition
1.0.2	04/05/2018	Ioannis Tamposis	Add Parallelization feature
1.0.3	24/09/2019	Ioannis Tamposis	Add HNN feature
1.0.4	22/04/2020	Ioannis Tamposis	Add Multiple Sequence Alignment feature

Contents

1. Introduction

JUCHMME, an acronym for Java Utility for Class Hidden Markov Models and Extensions is a tool developed for biological sequence analysis.

The overall aim of this work has been to develop a software tool that would offer a large collection of standard algorithms of Hidden Markov Models (HMMs), as well as, a number of extensions and to evaluate the software applied to various biological problems. The JUCHMME framework is characterized by:

Flexibility: Ease of use and customization for various problems. The user can create models of any architecture and any alphabet (DNA, protein or other), without requiring any programming capabilities (all required settings are defined in a configuration file).

Training methods: JUCHMME integrates a wide range of training algorithms for HMMs for labeled sequences. These kind of models are often referred to as “class HMMs” and are commonly trained using the Maximum Likelihood (ML) criterion to model within-class data distributions. The tool has been developed to support the Baum-Welch algorithm [1-3] and its extension that is necessary to handle labeled data [4]. Other alternatives are also supported, like the gradient-descent algorithm proposed by Baldi and Chauvin [5] and the Viterbi training (also known as “segmental K-means”) [6]. Additionally, the Conditional Maximum Likelihood (CML) criterion, which corresponds to discriminative training, is also supported. The CML training can be performed only with gradient based algorithms, and to this end a fast and robust algorithm for individual learning rate adaptation has been implemented [7]. The same algorithm is available for training the Hidden Neural Networks (HNN, see below).

Decoding: A wide range of decoding algorithms are integrated such as Viterbi, N–Best [8], posterior–Viterbi [9] and Optimal Accuracy Posterior Decoder [10]. Moreover, decoding of partially labeled data is offered with all algorithms in order to allow incorporation of experimental information [11].

Training Procedures: It contains built-in model creation and evaluation procedures, such as options for independent tests, self-consistency tests, jacknife tests, *k*–fold cross-validation and early stopping. All the prediction algorithms also incorporate appropriate reliability measures [12] and performance indices that have been widely used [13, 14] (such as the correlation coefficient, Q, or SOV).

HMM Extensions: To overcome standard HMM and class HMM limitations, a number of extensions have been developed such as segmental k–means (Viterbi training) for labeled sequences both for Maximum Likelihood (ML) [6] and for Conditional Maximum Likelihood (CML) training [15], Hidden Neural Networks (HNNs) [16], models that condition on previous observations [17] and a method for semi-supervised learning of HMMs that can incorporate labeled, unlabeled and partially-labeled data (semi–supervised learning) [18].

What HMMs are

Hidden Markov Models (HMMs) are probabilistic models. HMMs are generative models and in their basic formulation operate in an *unsupervised* manner, since they simply describe a finite mixture of multinomial distributions, where the mixture probabilities form a 1st order Markov chain. In this setting, during the *training* phase we maximize P(**x**|*θ*), which is the probability of the data given the model, whereas in the *decoding* phase we recover the hidden sequence of states that are most likely to have generated the data. In this manner, the only "supervision" needed is that of providing a reliable set of homologous sequences. When there is a need to compare different competing models, supervision is used indirectly, i.e. we train the different models separately and in the testing phase we simply choose the one with the highest probability (i.e. in a database search). In other applications, such as structure prediction, a sequence of labels (**y**) is tied to each observation sequence (**x**), corresponding to the different attributes that we wish to predict. In this case, we usually maximize P(**x**,**y**|*θ*), which is the joint probability of the sequences and the labels given the model, or P(**y**|**x**,*θ*), which is the probability of labels given the sequences and the model. These approaches typically correspond to a supervised learning procedure where each sequence **x** is accompanied by a complete sequence of well-defined labels **y**.

Applications of HMMs

The Hidden Markov Models (HMMs) are one of the most successful modeling approaches in speech recognition [3]. During the past two decades they were successfully applied on various tasks in computational molecular biology where they were proven to be useful for several problems in biological sequence analysis [2]. These include gene finding [19], multiple sequence alignment [20], prediction of signal peptides [21, 22], prediction of bacterial lipoproteins [23, 24], prediction of cell-wall sorting signals [25], prediction of protein secondary structure [26] and prediction of transmembrane protein topology [27, 28]. In several of these applications such as topology prediction of transmembrane proteins, HMMs have been found to perform significantly better compared to other sophisticated Machine-Learning techniques (e.g. Neural Networks or Support Vector Machines), as demonstrated in several evaluation studies [29-31].

A general definition of HMMs and an excellent tutorial introduction to their use has been written by Rabiner [3]. This shorthand usage is for convenience only. For a review of Class HMMs, see [19] and for a complete book on the subject of probabilistic modeling in computational biology, see [2].

2. Getting started

JUCHMME is a Java executable that can be run from the command line. JUCHMME is written in Java and requires a 32-bit or 64-bit Java runtime environment version 7 or later, freely available from http://www.java.org. The Windows and MacOS X installers contain a suitable Java runtime environment that will be used if a suitable Java runtime environment cannot be found on the computer.

Download the program from http://www.compgen.org/tools/juchmme or Github
https://github.com/pbagos/juchmme .

Compile:

```
javac -XDignore.symbol.file -sourcepath src/ -d ./bin src/hmm/Juchmme.java
javac -XDignore.symbol.file -sourcepath src/ -d ./bin src/hmm/RandomSeq.java

javac -XDignore.symbol.file -sourcepath src/ -d ./bin src/nn/Main.java
```

Libraries and other installation requirements:

JUCHMME includes a software library called JOONE (Java Object Oriented Neural Network) library (http://www.joone.org/) which is a Java framework to build and run AI applications based on neural networks, which it will automatically compile during its installation process. By default, JUCHMME does not require any additional libraries to be installed by you.

3. Arguments and Options

The *juchmme* program provides a list of command-line arguments and options.

– V: print JUCHMME version and exit

– a: the free emission parameter file (see section 4.4). This parameter file is **required**.

– e: the free transition parameter file (see section 4.3)

– i: the input sequence three-line file. This file stores the input sequences for decoding or training algorithms in a three-line format (see section 4.2).

– f: the input sequence FASTA file. This file stores the input sequences for decoding algorithms in FASTA format (see section 4.1).

– A: the input Multiple Sequence Alignment FASTA file. This file stores the input Multiple sequence Alignment (MSA) in for decoding algorithms in one line FASTA format (see section 4.10).

– m: the model file (see section 4.5). This parameter file is **required**.

– w: the HNN weights parameter file (see section 4.7)

– x: the HNN encoding file (see section 4.6)

– t: Training option

– c: the configuration file (see section 4.8)

– v cluster size: *k*–fold cross-validation mode using an integer larger than 0 for cluster size (for instance clusterSize=175)

– k number of clusters: *k*–fold cross-validation mode using an integer larger than 0 for k (for instance k=10)

– s: self-consistency test

– j: jacknife test

– p: show plot

– P: plot directory

By default, JUCHMME uses memory mapping to access its index files. If you intend to align a large number of files in a single run of JUCHMME, then it may be more efficient to have the program preload the complete index. To achieve this, use the command-line java option –Xmx**n**.

```
-Xmxn
```

Specify the maximum size, in bytes, of the memory allocation pool. This value must be a multiple of 1024 and greater than 2MB. Append the letter k or K to indicate kilobytes, or m or M to indicate megabytes. The default value is 64MB. The upper limit for this value will be approximately 4000m on Solaris 7 and Solaris 8 SPARC platforms and 2000m on Solaris 2.6 and x86 platforms, minus overhead amounts. Examples:

```
-Xmx83886080

-Xmx81920k

-Xmx80m
```

If you intend to use the extension of encoding (see section 5.2), the proposed method comprises of automatic conversion of initial symbols, according to the standard character encoding Unicode, using UTF-8 that is memory efficient and used by many operating systems and programming languages. To achieve this, use the command-line java option –Dfile.encoding=UTF-8.

```
java -Xmx4096m -Dfile.encoding=UTF-8 hmm/Juchmme -a ../tables/A_TMBB2 -e ../tables/E_TMBB2 -c ../conf/conf.tmbb -m ../models/tmbb.mdel -t ../input/barels_10LA.seq -d ../input/barrels14_seqs.txt
```

Random Sequence utility

Given a model, JUCHMME can generate a set of random sequences. This option can be useful for testing purposes. The user needs to provide the number of the sequences along with the transition, emission and configuration files for the given model.

```
java hmm/RandomSeq ../tables/A_TMBB2 ../tables/E_TMBB2 ../conf/conf.tmbb
../models/tmbb.mdel 100
```

4. Input/output files

4.1. The input sequence FASTA file

This file stores the input sequences for decoding algorithms in a FASTA-like format. The following is an example of an input sequence file:

```
>22 COXH_BOVIN
STALAKPQMRGLLARRLRFHIVGAFMVSLGFATFYKFAVAEKRKKAYADFYRNYDSMKDFEEMRKAGIFQSAK
```

4.2. The input sequence three-line file

This file stores the input sequences for decoding or training algorithms in a three-line FASTA format. The following is an example of an input sequence file:

```
>22 COXH_BOVIN
STALAKPQMRGLLARRLRFHIVGAFMVSLGFATFYKFAVAEKRKKAYADFYRNYDSMKDFEEMRKAGIFQSAK
IIIIIIIIIIIIIIIIMMMMMMMMMMMMMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

The first line has the FASTA header line, the second line has the observed sequence and the third line has the labels. If semi-supervised learning is used for training, the sequences in the input three-line file can contain unlabeled sequences (missing observations). The observation line followed by the character “-” for each amino acid like the following example:

```
>22 COXH_BOVIN
STALAKPQMRGLLARRLRFHIVGAFMVSLGFATFYKFAVAEKRKKAYADFYRNYDSMKDFEEMRKAGIFQSAK
-------------------------------------------------------------------------
```

or partially-labeled sequences like the following example:

```
>22 COXH_BOVIN
STALAKPQMRGLLARRLRFHIVGAFMVSLGFATFYKFAVAEKRKKAYADFYRNYDSMKDFEEMRKAGIFQSAK
----------------MMMMMMM-------MMMMMMMMMOOOOOOOOO-------------------------
```

4.3. The free transition parameter file

This file defines the free transition parameters and is required for setting up the transition probabilities of an HMM. All free transition parameters are stored in this file. We specify a free transition parameter for a 4-state HMM including begin (B) and end (E) states (see section 7.8) with the following format:

```
0.000 0.800 0.200 0.000 0.000 0.000
0.000 0.750 0.250 0.000 0.000 0.000
0.000 0.150 0.550 0.300 0.000 0.000
0.000 0.000 0.000 0.550 0.200 0.100
0.000 0.100 0.000 0.230 0.520 0.150
0.000 0.000 0.000 0.000 0.000 0.000
```

For a 4-state HMM model including begin (B) and end (E) states we need a file with 6 (lines) x 6 (columns). The six columns should be separated by a space or a tab character. The default value for the pseudo-probabilities is 0. In this free transition parameter file, each line represents one free transition parameter. Each line must sum to 1.

4.4. The free emission parameter file

This text file is required for setting up the emission probabilities of an HMM. We call a grouped set of emission probabilities which defines the emission probabilities for a state in the HMM a free emission parameter. All free emission parameters are stored in this file. We specify a free emission parameter for a 4-state HMM including begin (B) and end (E) states (see section 7.8) with the following format:

```
0.000 0.000 0.000 0.000
0.055 0.002 0.017 0.007
0.068 0.001 0.058 0.063
0.123 0.001 0.011 0.005
0.068 0.021 0.058 0.063
0.000 0.000 0.000 0.000
```

In the case of DNA for instance, an observed sequence is composed by a discrete set of 4 symbols, following the single-letter codes for the 4 nucleotides: A, C, G, T. For a 4-state HMM including begin (B) and end (E) states model we need a file with 4 (lines) x 6 (columns). The four columns should be separated by a space or a tab character.

4.5. The HMM model file

The models used by JUCHMME are described in files written in simple text format using straightforward conventions. It is easy to write, understand and modify them, or to create them using a separate program. This text file is required for setting up the model design. We specify a model file for an HMM with four states (B0, M1, M2, O1, O2, E0) two with label (M) and two with label (O). The model includes also begin (B) and end (E) states (see section 7.8).

```
# MODEL OPTIONS
MODEL=DEMO

ESYM=AGCT
OSYM=MmOoBE
PSYM=MOBE

#Model Unique Labels
transmLabels=M
inLabels=I
outLabels=O

#Model states and labels
STATE=B0 M1 M2 O1 O2 E0
OSTATE=B   M   m   O   o   E
PSTATE=B   M   M   O   O   E

#MODEL PRIOR for every esym
PRIOR = 0.077 0.018 0.058 0.066

# Distribution for each osym
# Each column must have a sum equal to 1
# osym    M     m    O    o   B   E
PRIOR1 = 0.97 0.97 0.97 0.95 0.0 0.0
PRIOR2 = 0.01 0.01 0.01 0.01 0.0 0.0
PRIOR3 = 0.02 0.02 0.02 0.04 0.0 0.0
```

OSYM represents the Alphabet of the observations.

OSYM represents the Alphabet of the “tied” states (parameter tying) (see section 5.2).

PSYM represents the Alphabet of the Labels.

STATE represents the model states.

OSTATE represents the “tied” states (parameter tying) (see section 5.2).

PSTATE represents the label for each state. Here we limit HMM models to one label for each state, which is probably the most useful approach for most known problems.

JUCHMME provides a very flexible way of integrating prior probabilities along the input sequences into the prediction process; a mapping between the labels of the states in the HMM and the different types of prior information has to be defined (see section 5.6).

4.6. The HNN Encoding parameter file

This text file is required for setting up the encoding of an HNN.

Array representation of Binary (SPARCE) Encoding Table. Refer to the below matrix for corresponding amino acids.

```
1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
```

Array representation of Blosum-62 Encoding Table. Refer to the below matrix for corresponding amino acids.

```
4    0    -2   -1   -2   0    -2   -1   -1   -1   -1   -2   -1   -1   -1   2    0    0    -3   -2
0    9    -3   -4   -2   -3   -3   -1   -3   -1   -1   -3   -3   -3   -3   -1   -1   -1   -2   -2
-2   -3   6    2    -3   -1   -1   -3   -1   -4   -3   1    -1   0    -2   0    -1   -3   -4   -3
-1   -4   2    5    -3   -2   0    -3   1    -3   -2   0    -1   2    0    0    -1   -2   -3   -2
-2   -2   -3   -3   6    -3   -1   0    -3   0    0    -3   -4   -3   -3   -2   -2   -1   1    3
0    -3   -1   -2   -3   6    -2   -4   -2   -4   -3   0    -2   -2   -2   0    -2   -3   -2   -3
-2   -3   -1   0    -1   -2   8    -3   -1   -3   -2   1    -2   0    0    -1   -2   -3   -2   2
-1   -1   -3   -3   0    -4   -3   4    -3   2    1    -3   -3   -3   -3   -2   -1   3    -3   -1
-1   -3   -1   1    -3   -2   -1   -3   5    -2   -1   0    -1   1    2    0    -1   -2   -3   -2
-1   -1   -4   -3   0    -4   -3   2    -2   4    2    -3   -3   -2   -2   -2   -1   1    -2   -1
-1   -1   -3   -2   0    -3   -2   1    -1   2    5    -2   -2   0    -1   -1   -1   1    -1   -1
-2   -3   1    0    -3   0    1    -3   0    -3   -2   6    -2   0    0    1    0    -3   -4   -2
-1   -3   -1   -1   -4   -2   -2   -3   -1   -3   -2   -2   7    -1   -2   -1   -1   -2   -4   -3
-1   -3   0    2    -3   -2   0    -3   1    -2   0    0    -1   5    1    0    -1   -2   -2   -1
-1   -3   -2   0    -3   -2   0    -3   2    -2   -1   0    -2   1    5    -1   -1   -3   -3   -2
1    -1   0    0    -2   0    -1   -2   0    -2   -1   1    -1   0    -1   4    1    -2   -3   -2
0    -1   -1   -1   -2   -2   -2   -1   -1   -1   -1   0    -1   -1   -1   1    5    0    -2   -2
0    -1   -3   -2   -1   -3   -3   3    -2   1    1    -3   -2   -2   -3   -2   0    4    -3   -1
-3   -2   -4   -3   1    -2   -2   -3   -3   -2   -1   -4   -4   -2   -3   -3   -2   -3   11   2
-2   -2   -3   -2   3    -3   2    -1   -2   -1   -1   -2   -3   -1   -2   -2   -2   -1   2     7
```

4.7. The HNN weights parameter file

This file defines the weights parameters and is required for setting up the weights probabilities of an HNN. All free weights parameters are stored in this file.

Weights are defined by the set of OSYM (without Begin and End States), the number of hidden neurons hidden layer (Configuration Setting) and the sliding window size multiplied by the set of symbols.

We specify a free weights parameter for a 4-osym-state HMM, 3 hidden neurons layer and a sliding window size 7 with the following format:

```
NEURAL      0
WTS12 -0.15696    1.14555     -8.59802    ...   16.01435
WTS12 -0.62859    0.85758     -0.07924    ...   -3.65333
WTS12 -5.31464    -55.84822   8.62123     ...   -33.08123
WTS23 -0.19443    -1.11150    0.91709     -0.42294
NEURAL      1
WTS12 -0.23986    0.10515     -21.70516   ...   10.46336
WTS12 -0.18695    20.63290    -21.92341   ...   -4.76647
WTS12 -0.59495    -30.60860   -16.51392   ...   -1.63771
WTS23 -0.00441    -0.039497   0.08402     -0.05086
NEURAL      2
WTS12 -7.64004    -56.26525   -27.66893   ...   15.18229
WTS12 -8.28003    -34.02406   -37.30982   ...   -29.11880
WTS12 0.257946    0.84665     3.00777     ...   9.91106
WTS23 1.794773    -3.73520    -2.055307   -12.45842
NEURAL      3
WTS12 -14.88240   24.10897    -10.74230   ...   -5.57302
WTS12 -2.84271    3.16543     33.28198    ...   32.04171
WTS12 -1.01723    -6.44938    30.87235    ...   18.17052
WTS23 0.38879     -2.49222    -1.77682    -2.89902
NEURAL      4
WTS12 -14.25349   10.14863    -84.25486   ...   -38.58804
WTS12 4.70765     -53.21236   92.12243    ...   7.30671
WTS12 -1.95006    -37.02785   39.10344    ...   14.11872
WTS23 0.67631     -2.58945    -7.75523    0.15810
```

4.8. The configuration file

A configuration file makes it possible to use most of the library’s algorithms without writing any line of code.

There are a number of options for specifying the output generated by the program.

TRAINING OPTIONS

RUN_CML: Use this option to specify if Conditional Maximum Likelihood (CML) method is to be used; it accepts two values, i.e. true/false (default value is *false*). CML implies Gradient (RUN_GRADIENT).

RUN_GRADIENT: Use this option to specify if Gradient method is to be used; it accepts two values, i.e. true/false (default value is *false*). If RUN_CML is true, then Gradient is used by default. If both CML and GRADIENT options are false, then the traditional Baum-Welch algorithm is used.

HNN: Use this option to specify if Hidden Neural Network (HNN) method is to be used; it accepts two values, i.e. true/false (default value is *false*)

ALLOW_BEGIN: Use this option to specify if Begin state is to be used; it accepts two values, i.e. true/false (default value is *true*)

ALLOW_END: Use this option to specify if End state is to be used; it accepts two values, i.e. true/false (default value is *true*)

RUN_ViterbiTraining: Use this option to specify if Viterbi Training Method is to be used; it accepts two values, i.e. true/false (default value is *false*). VITERBI can be used, as the method of computing expected counts, with all the previous combinations of CML and GRADIENT.

threshold: This option specifies a threshold score for terminating the training algorithm; the default value is *0.02*.

maxIter: This attribute specifies the maximum number of iterations for terminating the training algorithm; the default value is *200*.

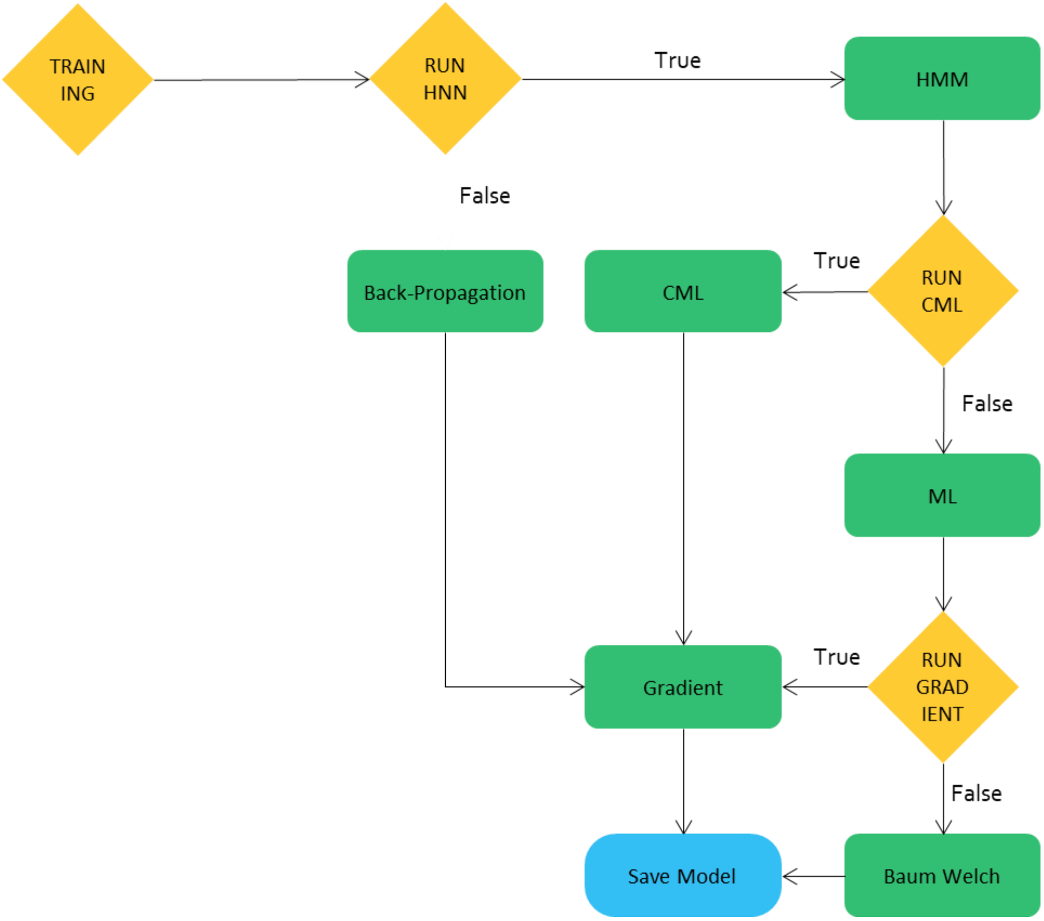


Fig. 1. The training workflow.

PROBABILITIES

TRANSITIONS: Use this option to specify which method is to be used to initialize transition probabilities; it accepts either of the following four values (see section 4.3).

- FILE, Initialize Transition probabilities by file (default value)
- RANDOM, Randomizing Transitions (*)
- UNIFORM, Uniformizing Transitions (*)
- VITERBI, Initialize Transition probabilities using Viterbi method (*)

(*) for all these methods, a valid transition probabilities matrix is needed, since the grammar of the HMM depends on it in order to determine the non-zero (allowed) transitions.

EMISSIONS: Use this option to specify which method is to be used to initialize emission probabilities; it accepts either of the following four values ((see section 4.4).

- FILE, Initialize Emission probabilities by file (default value)
- RANDOM, Randomizing Emissions
- UNIFORM, Uniformizing Emissions
- VITERBI, Initialize Emission probabilities using Viterbi method

WEIGHTS: Use this option to specify which method is to be used to initialize HNN weights; it accepts either of the following four values (see section 4.7).

- FILE, Initialize HNN weights by file (default value)
- RANDOM_NORMAL, Randomizing HNN weights
- RANDOM_UNIFORM, Randomizing HNN weights using the Normal method
- RPROP, Initialize HNN weights using the RPROP method
- BOOT, Initialize HNN weights using the Bootstrap method

Multithreaded parallelization for multicores

PARALLEL: Use this option to specify if multithreaded parallelization method is to be used; it accepts two values true/false (default value is *false*). If you specify false value, the program will run in serial mode.

allCPU: Use this option to specify if JUCHMME will use all available cores on your machine; it accepts two values true/false (default value is *false*).

nCPU: Use this option to specify the number of cores each JUCHMME process will use for computation; it accepts positive values 0 < *value* < Available Cores on your machine.

SEMI-SUPERVISED LEARNING OPTIONS (see section 5.1)

SSL_ENABLED: Use this option to specify if Semi-supervised learning method is to be used, it accepts two values, i.e. true/false (default value is false).

SSL_METHOD: Use this option to specify which Semi-supervised training method is to be used, it accepts either of the following two values

– 1: SSL, Self-training (default value)
– 2: GEM, Generalized EM

SSL_ADD_METHOD: Use this option to specify which method is to be used to add sequences. The option is defined by an integer; it accepts either of the following four values:

– 1: Use all (default value)
– 2: Use all, weighted by a (constant) value of λ<1
– 3: Use all, weighted by $P(\mathbf{y}^*|\mathbf{x}^u,\theta)$
– 4: Use most confident examples, those with reliability less than a given threshold

SSL_USING_METHOD: Use this option to specify which decoding method is to be used. The option is defined by an integer; it accepts either of the following four values:
– 1: VITERBI
– 2: NBEST [8]
– 3: Posterior–Viterbi (POSVIT) [9]
– 4: Optimal Accuracy Posterior Decoder (PLP) [10] (default value)

SSL_THRESHOLD: This option specifies a threshold score for terminating the semi-supervised training algorithm (default value is *0.000002*)

SSL_maxIter: This attribute specifies the maximum number of iterations for terminating the semi-supervised training algorithm (default value is *200*).

SSL_relscore: This attribute specifies the optimal threshold used by method 4 (default value is *0.95*)

SSL_WEIGHT: This attribute specifies the (constant) value of λ used by method 2 (default value is *0.2*).

EXTENDED ENCODING OPTIONS (see section 6.2)

PAST_OBS_EXTENSION: Use this option to specify if Extending Encoding Method is to be used; it accepts two values, i.e. true/false (default value is *false*).

ENCODE_TYPE: Use this option to specify the Encoding Scheme. The option is defined by an integer; it accepts either of the following four values. If you specify 0 as value, the program will run with your own Encoding scheme taking into account the parameters *GROUP_SYMBOLS* and *GROUPING*:

– 1 (default value): An Encoding with 40 (20x2) symbols depending on whether the previous residue is hydrophobic (A, F, H, I, L, M, V, W, Y) or non-hydrophobic (C, D, E, G, K, N, P, Q, R, S, T). (Encoding 1).
– 2: An Encoding with 80 (20x4) symbols depending on whether the previous residue is: Hydrophobic–Aromatic (F, H, Y, W), Hydrophobic–non-Aromatic (A, I, L, M, V, G), non-Hydrophobic–Charged (D, E, K, R), non-Hydrophobic–Polar (C, N, P, Q, S, T). (Encoding 2).
– 3: An Encoding with 160 (20x8) symbols depending on whether the previous residue is: Hydrophobic–Small (A, G), Polar–Special (P, C), Polar–OH (S, T), Polar–NH (N,

Q), Charged–Negative (D, E), Charged–Positive (K, R), Hydrophobic–Huge (I, L, M, V) and Hydrophobic–Aromatic (F, H, Y, W). (Encoding 3).

- 4: An Encoding with 400 (20x20) symbols that takes into account all possible dipeptide combinations. (Encoding 4).

GROUP_SYMBOLS: Use this option to specify the number of groups. The option is defined by a String. For instance, to define an encoding with two groups, depending on whether the previous residue is hydrophobic (A, F, H, I, L, M, V, W, Y) or non-hydrophobic (C, D, E, G, K, N, P, Q, R, S, T), use 0 for hydrophobic and 1 for non-hydrophobic (see section 6.2).

GROUPING: Use this option to specify in which group corresponds each letter of the alphabet. The option is defined by a String. For instance, to define an encoding with two groups, depending on whether the previous residue is hydrophobic (A, F, H, I, L, M, V, W, Y) or non-hydrophobic (C, D, E, G, K, N, P, Q, R, S, T), use value 10001011011000000111 (see section 6.2).

PAST_OBS_NO: Use this option to specify the number of previous observations. The option is defined by an integer (default value is *1*).

DYNAMIC OPTIONS

These parameters control the dynamic programming algorithm used for imposing constraints in Posterior decoding prediction. The algorithm is based on [32-35], but it is considered obsolete.

MINHLEN: Use this option to specify the minimum length of predicted strands (default value is *7*).

MINLLEN: Use this option to specify the minimum length of predicted loop (default value is *1*).

MAXHLEN: Use this option to specify the maximum length of predicted strands (default value is *17*).

MAXNSTRAND: Use this option to specify the maximum number of predicted strands (default value is *32*).

MINSSC: Use this option to specify the minimum score of predicted strands (default value is *3*).

STRDIV: Use this option to specify the minimum sequence length required for one predicted strand (default value is *9*).

Refine OPTIONS (see section 5.7)

FLANK: Use this option to specify a flanking region (i.e. number of residues in each direction of the end of a membrane-spanning helix).

REFINE: Use this option to specify if Refine method is to be used; it accepts two values, i.e. true/false (default value is *false*).

ML_INIT: Use this option to specify if the starting output model would be CML with ML;, it accepts two values, i.e. true/false (default value is *false*).

<u>DECODING OPTIONS</u> (see section 5.4)

VITERBI: Use this option to specify if Viterbi Decoding method is to be used; it accepts two values, i.e. true/false (default value is *true*)

NBEST: Use this option to specify if NBest Decoding method is to be used; it accepts two values, i.e. true/false (default value is *false*) [8]

DYNAMIC: Use this option to specify if Dynamic Decoding method is to be used; it accepts two values, i.e. true/false (default value is *false*).

POSVIT: Use this option to specify if Posterior–Viterbi Decoding method is to be used; it accepts two values, i.e. true/false (default value is *false*) [9].

PLP: Use this option to specify if Optimal Accuracy Posterior Decoder Decoding method is to be used; it accepts two values, i.e. true/false (default value is *true*) [10].

All decoding methods can be used in conjunction.

CONSTRAINT: Use this option to specify if constraint method is to be used; it accepts two values, i.e. true/false (default value is *false*) (see section 5.5).

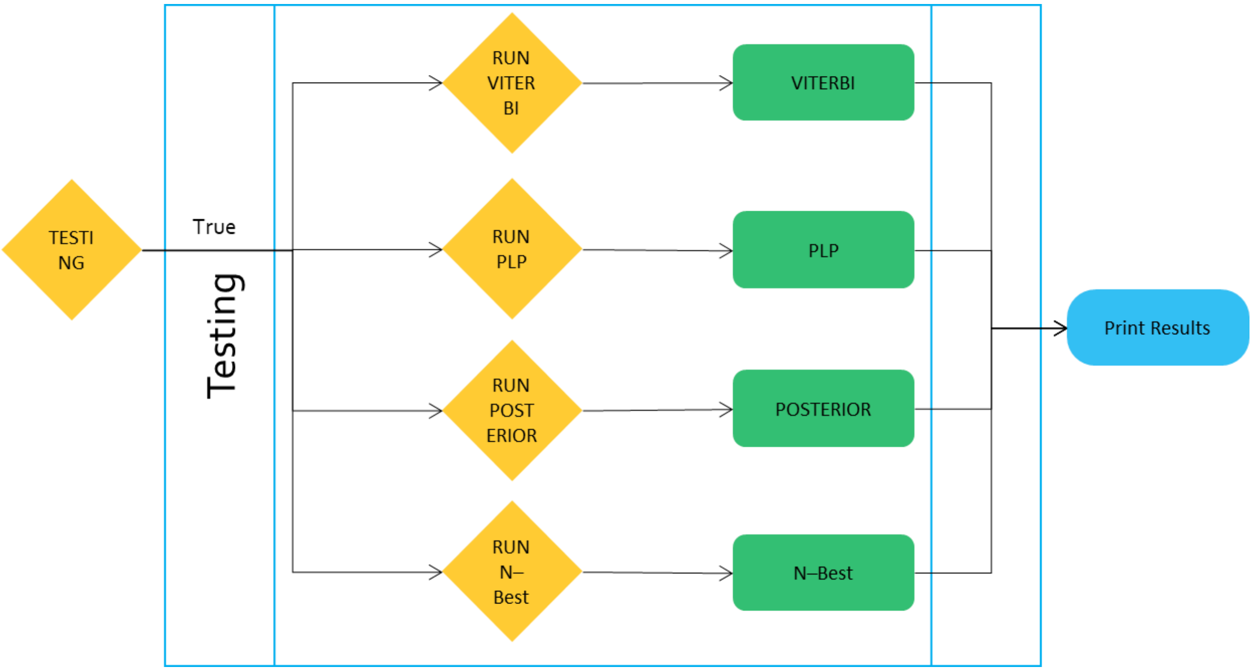


Fig. 2. The training workflow.

<u>EARLY STOPPING</u>

EARLY: Use this option to specify if the early stopping method is to be used; it accepts two values, i.e. true/false (default value is *false*).

CUSTOM_STOP: Use this option to specify a weight factor for Early Stopping method; the default value is *0.0*.

NTRAIN: This attribute specifies the number of sequences for training in the early stopping method (the default value is 15).

NROUND: This attribute specifies the maximum number of iterations for running the early stopping method (default value is *5*).

ITER: This attribute specifies the maximum number of iterations that start running the early stopping training algorithm (default value is *2*).

GRADIENT DESCENT OPTIONS

As presented in [7], JUCHMME supports two algorithms that use individual learning rate adaptation. If both options are set to TRUE then the algorithm #2 is used. This is a version of the RPROP algorithm [36] for individual learning rate adaptation, originally proposed for Neural Networks. If RPROP is false and SILVA is true, then the algorithm #1 is used, which uses an individual learning rate (a variant of the Silva and Almeida algorithm). The difference of the two algorithms lies in the fact that in RPROP the magnitude of the partial derivative is neglected and only its sign is used. This method seems to perform best. When both options are set to FALSE, standard gradient descent is applied. When RPROP is true and SILVA is false, a variant of the Manhattan method is applied, but this method is neither tested, nor is it expected to work well. For standard gradient descent we use learning rates, ranging between 0.001 to 0.1 for both emission (*kappaE*) and transition probabilities (*kappaA*), whereas the same values were used for the initial parameters for every parameter of the model. Finally, for setting the minimum and maximum allowed learning rates we used *kappaAmin*, *kappaEmin* equal to 10^{-20} and *kappaAmax*,*kappaEmax* equal to 10.

RPROP: Use this option to specify the training method. It is enabled if RUN_GRADIENT method is used (ML or CML) and it accepts two values, i.e. true/false (default value is *true*). The option works in combination with SILVA (*).

SILVA: Use this option to specify the training method. It is enabled if RUN_GRADIENT method is used (ML or CML) and it accepts two values, i.e. true/false (default value is *true*). The option works in combination with RPROP (*).

momentum: To avoid to get stuck in a local minima, we use a momentum term in the objective function, which is a value between 0 and 1 that increases the size of the steps taken towards the minimum by trying to jump from a local minima.

kappaA: Use this option to specify the learning rate of Transitions (default value is *0.01D*).

kappaAmin: Use this option to specify the minimum value of learning rate of Transitions, in case an individual learning rate training method is used (default value is *1e-20*).

kappaAmax: Use this option to specify the maximum value of learning rate of Transitions, in case an individual learning rate training method is used (default value is *1.0*).

kappaE: Use this option to specify the learning rate of Emissions (default value is *0.01D*).

kappaEmin: Use this option to specify the minimum value of learning rate of Emissions, in case an individual learning rate training method is used (default value is *1e-20*).

kappaEmax: Use this option to specify the maximum value of learning rate of Emissions, in case an individual learning rate training method is used (default value is *1.0*).

NPLUS: Use this option to specify a value for increasing factors (default value is *1.2*).

NMINUS: Use this option to specify a value for decreasing factors (default value is *0.5*).

PRIOR OPTIONS (see section 5.6)

NOISE_TR: Use this option to specify if prior information is to be added in transitions; it accepts two values, i.e. true/false (default value is *true*).

NOISE_EM: Use this option to specify if prior information is to be added in emissions; it accepts two values, i.e. true/false (default value is *true*).

PRIOR_TRANS: Use this option to specify a prior probability in [0:1] of Transitions (default value is *0.001*).

HNN OPTIONS (see section 6.3)

windowLeft: Use this option to specify the sequence window size on the left, default value is *3*.

windowRight: Use this option to specify the sequence window size on the right, default value is *3*.

nhidden: This attribute specifies the number of hidden neurons of the hidden layer, the default value is *3*.

ADD_GRAD: Use this option to specify a weight factor for error in the Gradient Descent Method (default value is *0.0*).

DECAY: Use this option to specify a weight decay that causes the weights to exponentially decay to zero (default value is *0.001*). A different way to constrain a network, and thus decrease its complexity, is to limit the growth of the weights through some kind of weight decay. It should prevent the weights from growing too big unless it is really necessary. It can be imposed by adding a term to the cost function that penalizes large weights [37].

hiddenLayerFunction: Use this option to specify which hidden layer activation function is to be used.

- 1: Sigmoid Function.
- 2: Sigmoid Modified Function (default value).
- 3: Tanh.

BOOTSTRAP OPTIONS (HNN ONLY)

BOOT: Use this option to specify the maximum number of iterations to weights initialization by Bootstrap Method (the default value is *0*).

STDEV: Use this option to specify the Standard Deviation (the default value is *1.5*).

RANGE: Use this option to specify a double number to initialize the pseudo-random weights (the default value is *5.0*).

SEED: Use this option to specify a long number to initialize the pseudo-random weights (the default value is *568381*).

WEIGHT_RAND: Use this option to add noise to weights initialization (the default value is *0.0*).

WEIGHT_TIME: Use this option to specify if a seed value is to be used; it accepts two values, i.e. true/false (default value is *false*).

RpropNN OPTIONS (HNN ONLY)

numberOfCycles: This attribute specifies the maximum number of iterations for terminating the training algorithm (the default value is *150*).

doCrossVal: Use this option to specify if Cross Validation method is to be used; it accepts two values, i.e. true/false (default value is *false*).

crossValIter: This attribute specifies the maximum number of iterations for terminating the Cross-Validation training procedure (the default value is *5*).

minGEdiff: This attribute specifies the maximum number of error between two iterations to terminate the training algorithm (the default value is *0*).

globalError: This attribute specifies the global Error function, it accepts two values, i.e. RMSE/CE (default value is *RMSE*)

initialDelta: Use this option to specify the learning rate (default value is *0.1*).

maxDelta: Use this option to specify the maximum delta value (default value is *50*).

minDelta: Use this option to specify the minimum delta value (default value is *1e-6*).

etaInc: Use this option to specify the incremental learning factor/rate (default value is *1.2*).

etaDec: Use this option to specify the decremental learning factor/rate (default value is *0.5*).

4.9. The Output text file

JUCHMME provides a simple output text file for the free transition parameters (see section 4.3), a simple output text file for the free emission parameters (see section 4.4) and a simple output text file for the HNN weights parameters (see section 4.7).

JUCHMME also provides a simple output text file for storing the results of sequence decoding or training. The first section is the header that tells you what program you ran, on what, and with which options.

```
JUCHMME :: Java Utility for Class Hidden Markov Models and Extensions
Version 2.1; September 2018
Copyright (C) 2018 Pantelis Bagos
Freely distributed under the GNU General Public Licence (GPLv3)
--------------------------------------------------------------------------
Preparing System Arguments
--------------------------------------------------------------------------
Preparing System Model (../models/demo.mdel)
Using model DEMO
Model has 6 states.
--------------------------------------------------------------------------
Preparing System Configuration (../conf/conf.demo)
Multithreaded parallelization = true
Processors : 10
Ka = 0.01      min = 1.0E-20  max = 1.0
Ke = 0.01      min = 1.0E-20  max = 1.0
momentum = 0.0
RPROP = true   SILVA =true    Weight Decay = 0.001
PRIOR_TRANS= 0.001
--------------------------------------------------------------------------
Preparing Sequences
100 sequences in file ../input/demoSet.seq
100 sequences are Labeled
--------------------------------------------------------------------------
Initialize Transition Probabilities By File
Initialize Emission Probabilities By File
--------------------------------------------------------------------------
```

The second section is the training, which starts with the label TRAINING and the training procedure that can be one of Self Consistency, Cross-Validation or JackKnife. Then, all training steps according to the system configurations are shown.

```
TRAINING - Self Consistency
        Computing Forward+Backward (Clumped)
        -----------------------------------------------------------------------------------------
        *****************************************************************************************
1       log likelihood = -2275.8711098899776
        Computing expected counts
        .........................................................................................
        Updating transitions and emissions using Baum-Welch
E zero at k=0
A zero at k=5
E zero at k=5
        Computing Forward+Backward (Clumped)
        -----------------------------------------------------------------------------------------
        *****************************************************************************************
2       log likelihood = -2614.1851848861643          diff = 338.3140749961867
        Computing expected counts
        .........................................................................................
        Updating transitions and emissions using Baum-Welch
E zero at k=0
A zero at k=5
E zero at k=5
```

The third section is the TESTING/DECODING. First, JUCHMME reports the sequence details such as sequence identifier (ID), sequence (SQ) and sequence observation (OB). Next, results are generated for each algorithm that the user defined in the configuration file. For each algorithm, we use a different set of labels and, more specifically, a different label at the beginning of each line. For instance, in the case of the VITERBI algorithm we use the labels (VS, VR, VP, where S stands for Score, R stands for Reliability and P stands for Path), in the case of the Optimal Accuracy Posterior Decoder algorithm the respective labels (LS, LR, LP), for Posterior-Viterbi we have PS, PR and PP, whereas for N-best we have NS, NR, NP.

```
TESTING
ID: >SEQ_82
SQ: ACAAACGAAATCCACACAAAC
OB: MMMMMMMMMMMMMOOOOOOOO
CC:     lng = 21.0     logodds = 27.676880823281603  (-logprob/lng) = 1.4170089237786447
VS: -3.143084398920788
VR: 0.5443280039582037
VP: MMMMMMOOOOOOOOOOOOOOO
PS: 19.244528764979968
PR: 0.7914446048321854
PP: MMMMMMMMMMMMMOOOOOOOO
LS: 32.567814447386866
LR: 0.7914446048321854
LP: MMMMMMMMMMMMMOOOOOOOO
NS: -1.8184147700199027
NR: 0.7914446048321854
NP: MMMMMMMMMMMMMOOOOOOOO
```

The fourth section is the statistics. JUCHMME prints statistical results for each decoding algorithm which starts with the label identified by its name (see section 5.8). For measures of accuracy in case of membrane protein prediction methods we used the fraction of the correctly predicted residues in a two-state mode (Q2), the fraction of proteins with correctly predicted topologies, the segments over-lap measure (SOV) and the Mathews correlation coefficient (MCC) that summarizes in a single measure true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). In other cases, users can use decoding output to implement their measures.

```
VITERBI:

Q2:0.845       Qa:0.737       Qna:0.916      Pa:0.855       Pna:0.839      Qfas:0.827
Ca:0.673       SM:0.885       TP:596  FP:37  FN:122  Correct Top:10 Correct Ori:10 Avg SOV:0.787

PLP:

Q2:0.852       Qa:0.772       Qna:0.906      Pa:0.846       Pna:0.856      Qfas:0.839
Ca:0.690       SM:0.904       TP:627  FP:47  FN:91   Correct Top:14 Correct Ori:14 Avg SOV:0.807
```

Finally, JUCHMME reports the Execution time.

```
Execution time = 561 miliseconds
Execution time = 0.561 seconds
```

4.10. The input Multiple Sequence Alignment FASTA file

This file stores the input Multiple Sequence Alignment for decoding algorithms in one line FASTA format. The following is an example of an input sequence file:

```
>B1L914|3dinE
MKTF-FL-IVHTI-ISVAL-I-Y-MVQVQ-M-S-K---F--S--E-L-----G-----G---AF----G-----S--
--G-----G-------L-----H----T--V---F----G---R----R--K--G--L---D----T---G-G-K--
-I---T-LVLSVLFFVSCV--VT--AF-V-L-TR-------
>A0A2A4PUF1/1-81 [subseq from] A0A2A4PUF1
MQTI-LL-LIHVV-IAVAL-V-G-LVLLQ-H-G-K---G--A--D-A-----G-----A---AF----G-----S--
--G-----A-------S-----Q----T--V---F----G---S----Q--G--S--G---S----F---L-T-R--
-A---T-GILATVFFVTSL--VL--AY-L-S-XGQVTNS--
>A0A1H8RXY5/1-83 [subseq from] A0A1H8RXY5
MAQI-VL-IFHVV-IAIAL-V-V-LVLLQ-H-G-K---G--A--D-A-----G-----A---AF----G-----S--
--G-----A-------S-----S----T--V---F----G---A----R--G--S--A---T----F---L-S-R--
-I---T-AMLAAGFFITSL--TL--AM-F-A-SRDAGPGSV
```

5. Supported features

JUCHMME provides a simple and user-friendly text input file format for configuration, model design and model probabilities.

5.1. Determine the Initial Probabilities of the HMM

JUCHMME provides a flexible functionality to parameterize/initialize transition, emission and weight probabilities. Transition probabilities are required to be provided as input since they describe the model itself. In the initial transition probabilities table, non-zero entries define the allowed transitions between states and thus define the model architecture (a transition probability of zero cannot be undone). Emission probabilities are required in case an HMM is to be used and are obsolete in case an HNN is to be used. The user can specify in the configuration settings if it needs to randomize or to uniformize either transition or emission probabilities. Moreover, the user can choose to initialize probabilities using the Viterbi algorithm on the training data (this is the suggested option for real-life applications).

Configuration Settings

```
#PROBABILITIES
# FILE, RANDOM, UNIFORM, VITERBI
TRANSITIONS=FILE
# FILE, RANDOM, UNIFORM, VITERBI
EMISSIONS=FILE
# FILE, RANDOM_NORMAL, RANDOM_UNIFORM, RPROP, BOOT
WEIGHTS=RPROP
```

5.2. Tying

Another feature is called tying (or parameter sharing), which is used extensively in speech recognition (the same technique is called ‘weight sharing’ for neural networks). Tying of two states means that the emission probabilities are always identical in the two states. To keep the number of parameters at a reasonable level, one could apply tying as commonly done in context-dependent modeling with standard HMMs. For example, if the M symbol is modeled as a left context part (M) and a core part (m), one could tie the match networks between the "core states" across all sub-models (m, M) for the same symbol (M). This would largely decrease the number of parameters (see section 4.5).

5.3. Training

JUCHMME provides a variety of computationally efficient parameter training algorithms: linear-memory Baum-Welch training means both for Maximum Likelihood (ML) and for Conditional Maximum Likelihood (CML), linear-memory Viterbi training, linear-memory posterior sampling training and Gradient Descent for Conditional Maximum Likelihood (CML).

Traditionally, HMM training is performed using the Baum-Welch algorithm [1-3] which is a special case of the Expectation-Maximization (EM) algorithm for missing data [4]. Missing data in this case is the path π (i.e. the sequence of states), since, if we knew the exact path, the Maximum Likelihood (ML) estimates could be easily derived by counting the observed transitions and emissions. An alternative to the Baum-Welch algorithm, even though not widely used, is the gradient-descent algorithm proposed by Baldi and Chauvin [5]. In any case, in these models, maximization of the likelihood corresponds to an unsupervised learning problem.

In other biological sequence analysis problems, where we want to classify various segments along the sequence, we often use labeled sequences for training. In such cases, each amino acid sequence **x** is accompanied by a sequence of labels **y** for each position i in the sequence ($\mathbf{y}=y_1, y_2, \ldots, y_L$). Consequently, we declare a new probability distribution, the probability $\delta_k(y_i=c)$ of a state k having a label c. In most applications, this probability is just a delta-function, since a particular state is not allowed to match more than one label. Krogh proposed a simple modification of forward and backward algorithms in order to incorporate information from labeled data [19]. The likelihood to be maximized in such situations is the joint probability of the sequences (**x**) and the labels (**y**) given the model, in which the summation is done only over those paths Π_y that are in agreement with the labels **y**. This typically corresponds to a *supervised learning procedure*. With the use of labeled sequences, we can also perform a kind of discriminative training, with a criterion known as Conditional Maximum Likelihood (CML). Furthermore, there is no EM algorithm for training and one has to use general gradient-based methods [11].

Configuration Settings

```
# TRAINING OPTIONS
RUN_CML=false
RUN_GRADIENT=false
HNN=false
ALLOW_BEGIN=true
ALLOW_END=true
RUN_ViterbiTraining=false
threshold=0.02
maxIter=200
```

Forward and Backward algorithms

The standard Forward (1) and Backward (2) algorithms are employed in the likelihood calculations in HMMs.

$$\begin{aligned}
&\forall k \neq B, i = 0 : f_B(0) = 1,\ f_k(0) = 0 \\
&\forall 1 \leq i \leq L : f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl} \qquad (1) \\
&P(x|\theta) = \sum_k f_k(L) a_{kE}
\end{aligned}$$

$$\begin{aligned}
&\forall k \neq B, i = L : b_B(L) = a_{kE} \\
&\forall 1 \leq i \leq L : \mathrm{b}_k(i) = \sum_k a_{kl} e_l(x_i + 1) b_l(i-1) \qquad (2) \\
&P(x|\theta) = \sum_l a_{Bl} e_l(x_1) b_l(1)
\end{aligned}$$

5.4. Decoding

JUCHMME provides a variety of Algorithms for generating predictions by sequence decoding using the standard Viterbi (VITERBI) [2] algorithm and also more advanced techniques such as the N–Best (NBEST) [8], Posterior-Viterbi decoding (POSVIT) [9] and Optimal Accuracy Posterior Decoder (PLP) [10].

Viterbi algorithm

The Viterbi algorithm is a dynamic programming algorithm that, contrary to the Forward and Backward algorithms, finds the likelihood of the most probable path of states and not the total probability of all paths. With a backtracking step, the algorithm finally recovers the most probable path. The algorithm is conceptually similar to the Forward algorithm, where the consecutive summations are replaced by maximizations:

$$\begin{aligned}
&\forall k \neq B, i = 0 : \mathrm{u}_B(0) = 1, \mathrm{u}_k(0) = 0 \\
&\forall 1 \leq i \leq L : \mathrm{u}_l(i) = e_l(x_i) \max_k \left\{ u_k(i-1) a_{kl} \right\} \\
&P\left(x, \pi^{\max} \middle| \theta\right) = \max \left\{ u_k(L) a_{kE} \right\}
\end{aligned} \tag{3}$$

1-best algorithm

The 1-best decoding is a modification of the N-best decoding method, proposed earlier for speech recognition [38]. It is a heuristic algorithm that tries to find the most probable path of labels (y^{max}) of a sequence instead of the most probable path of states. For each position i in the sequence, it keeps track of all the possible active hypotheses h_{i-1} that consist of all the possible sequence of labels up to that point. Afterwards, for each state l, it propagates these hypotheses, appending each one of the possible labels y_i, and picks up the best, until the end of the sequence. In contrast to the Viterbi algorithm, 1-best does not need a traceback procedure:

$$\begin{aligned}
&i = 1 : \gamma_l(h_1) = a_{Bl} e_l(x_1) \\
&\forall 1 \leq i \leq L : \gamma_l(h_i \gamma_i) = e_l(x_i) \sum_k \gamma_k(h_{i-1}) a_{kl} \\
&P\left(x, y^{\max} \middle| \theta\right) = \sum_k \gamma_k(h_L) a_{kE}
\end{aligned} \tag{4}$$

The main drawback of 1-best is that the computation time is significantly longer than Viterbi, However, the memory requirement is only weakly dependent on sequence length, since it depends on the number of states in the model. The low memory requirement is particularly useful when parsing very long sequences, where the simple Viterbi algorithm requires more memory.

Posterior-Viterbi decoding (POSVIT)

Fariselli and co-workers have provided a decoding algorithm that combines elements of the Viterbi and the posterior decoding [9]. The Posterior-Viterbi decoding algorithm performs essentially a Viterbi-like decoding, using the Posterior probabilities instead of the emissions and the allowed paths given by the delta function instead of the transitions.

$$
\begin{aligned}
&\forall k \neq B, i = 0 : \mathrm{u}_B(0) = 1, \mathrm{u}_k(0) = 0 \\
&\forall 1 \leq i \leq L : \mathrm{u}_l(i) = P\left(\pi_i = l \underset{k}{\Big|} x, \theta\right) \max\left\{u_k(i-1)\delta(k,l)\right\} \\
&P\left(x, \pi^{\max} \middle| \theta\right) = \max\left\{u_k(L)\delta(k,E)\right\} \\
&\delta(k,l) = \begin{cases} 1, if a_{kl} > 0 \\ 0, \text{otherwise} \end{cases} \\
&\pi^{PV} = \arg\max_{\pi} \prod_{i=1}^{L} \delta(\pi_i, \pi_{i+1}) P\left(\pi_i \middle| x\right)
\end{aligned}
\tag{5}
$$

Optimal Accuracy Posterior Decoder algorithm (PLP)

Käll and coworkers presented a very similar algorithm, the Optimal Accuracy Posterior Decoder which requires class HMMs [10].

$$
\begin{aligned}
&\forall k \neq B, i = 0 : A_B(0) = 0, A_k(0) = -\infty \\
&\forall 1 \leq i \leq L : A_l(i) = P\left(\gamma_i = c^l \middle| x, \theta\right) + \max_k\left\{A_k(i-1)\delta(k,l)\right\} \\
&P\left(x, \pi^{OAPD} \middle| \theta\right) = \max_k\left\{A_k(L)\delta(k,E)\right\} \\
&\delta(k,l) = \begin{cases} 1, if a_{kl} > 0 \\ 0, \text{otherwise} \end{cases} \\
&\pi^{OAPD} = \arg\max_{\pi} \sum_{i=1}^{L} \left\{ \delta(\pi_i, \pi_{i+1}) \left(\sum_k P\left(\pi_i \middle| x\right) \lambda_k(c) \right) \right\}
\end{aligned}
\tag{6}
$$

Configuration Settings

```
# DECODING OPTIONS
VITERBI=true
NBEST=false
DYNAMIC=false
POSVIT=false
PLP=true
```

5.5. Constrained predictions

This is a simple method that allows incorporation of prior topological information when performing a constrained prediction [11, 28] .

Configuration Settings

```
CONSTRAINT=false
```

We will define the concept of the *Information*, ω that consists of 1≤*r*≤*L* residues of the sequence **x**, of which we know the experimentally determined labels, and thus the (*a priori*) labelling ω_i: ω = ω_1, ω_2, ...,ω_r . According to this terminology, the set of residues with *a priori* known labels ω_r, is a subset of the set *I* = (1,2, ...,*L*) defined by the residues of the sequence. The labels ω_i should belong to the same set of labels defined in the model architecture.

Modified Forward algorithm

$$\begin{aligned}
&\forall k \neq B, i = 0 : f_B^{\omega}(0) = 1, f_k^{\omega}(0) = 0 \\
&\forall 1 \leq i \leq L : f_l^{\omega}(i) = e_l(x_i) \sum_k f_k^{\omega}(i-1) a_{kl} \\
&P(x, \omega | \theta) = \sum_k f_k^{\omega}(L) a_{kE}
\end{aligned} \tag{7}$$

Modified Backward algorithm

$$\begin{aligned}
&\forall k \neq B, i = L : b_B^{\omega}(L) = a_{kE} \\
&\forall 1 \leq i \leq L : \mathrm{b}_k^{\omega}(i) = \sum_k a_{kl} e_l(x_i + 1) b_l^{\omega}(i-1) \\
&P(x, \omega | \theta) = \sum_l a_{Bl} e_l(x_1) b_l^{\omega}(1)
\end{aligned} \tag{8}$$

Modified Viterbi algorithm

$$\begin{aligned}
&\forall k \neq B, i = 0 : \mathrm{u}_B^{\omega}(0) = 1, \mathrm{u}_k^{\omega}(0) = 0 \\
&\forall 1 \leq i \leq L : \mathrm{u}_l^{\omega}(i) = e_l(x_i) \max_k \left\{ u_k^{\omega}(i-1) a_{kl} \right\} \\
&P\left(x, \omega, \pi^{\omega} \middle| \theta\right) = \max \left\{ u_k^{\omega}(L) a_{kE} \right\}
\end{aligned} \tag{9}$$

Modified 1-best algorithm

$$\begin{aligned}
&i = 1 : \gamma_l^{\omega}(h_1) = a_{Bl} e_l(x_1) \\
&\forall 1 \leq i \leq L : \gamma_l^{\omega}(h_i \gamma_i) = e_l(x_i) \sum_k \gamma_k^{\omega}(h_{i-1}) a_{kl} \\
&P\left(x, y^{\omega} \middle| \theta\right) = \sum_k \gamma_k^{\omega}(h_L) a_{kE}
\end{aligned} \tag{10}$$

Modified Posterior-Viterbi algorithm

$$\begin{aligned}
&\forall k \neq B, i = 0 : \mathrm{u}_B^{\omega}(0) = 1, \mathrm{u}_k^{\omega}(0) = 0 \\
&\forall 1 \leq i \leq L : \mathrm{u}_l^{\omega}(i) = P\left(\pi_i = l \middle| x, \omega, \theta\right) \max_k \left\{ u_k^{\omega}(i-1) \delta(k, l) \right\} \\
&P\left(x, \omega, \pi^{PV,\omega} \middle| \theta\right) = \max \left\{ u_k^{\omega}(L) \delta(k, E) \right\}
\end{aligned} \tag{11}$$

Modified Optimal Accuracy Posterior Decoder algorithm

$$\begin{aligned}
&\forall k \neq B, i = 0 : A_B(0) = 0, A_k(0) = -\infty \\
&\forall 1 \leq i \leq L : A_l(i) = P\left(\gamma_i = c^l \middle| x, \omega, \theta\right) + \max_k \left\{ A_k^{\omega}(i-1)\delta(k,l) \right\} \qquad (12) \\
&P\left(x, \omega, \pi^{OAPD} \middle| \theta\right) = \max_k \left\{ A_k^{\omega}(L)\delta(k,E) \right\}
\end{aligned}$$

5.6. Prior probabilities

JUCHMME provides a very flexible way for integrating prior probabilities along the input sequences into the prediction process; a mapping between the labels of the states in the HMM and the different types of prior information has to be defined. This was implemented by adding noise to the model parameters, and then decreasing the level of this noise by prior value per iteration. For transitions, an optional Boolean attribute defines (*PRIOR_TRANS*) whether the annotation label set accepts a prior probability in [0:1]. For emissions, an optional Boolean for each symbol is defined in Model Settings. Users can define any distribution for each state in an HMM model. The system takes care of the correct normalization of the prior probabilities. This means that the sum of the prior probabilities for mutually exclusive annotation labels in one annotation label set assigned on the same interval of sequence positions has to be at most 1 (and exactly 1, if the annotation labels of one set cover all possibilities).

New_A = A + PRIOR_TRANS * A_initial

New_E = E x PRIOR1 + E_initial x PRIOR2 + PRIOR x PRIOR3

```
Exception in thread "main" java.lang.Exception: ERROR: Zero probability at
position 1. Symbol: G Obs: I.
```

Configuration Settings

```
#PRIOR OPTIONS
NOISE_TR=true
NOISE_EM=true
PRIOR_TRANS=0.001
```

Model Settings

```
#MODEL PRIOR for every symbol (esym)
PRIOR = 0.077 0.018 0.058 0.066

# Distribution for each state (osym)
# Each column must have a sum equal to 1
# osym     M    m    O    o   B   E
PRIOR1 = 0.97 0.95 0.97 0.97 0.0 0.0
PRIOR2 = 0.01 0.01 0.01 0.01 0.0 0.0
PRIOR3 = 0.02 0.04 0.02 0.02 0.0 0.0
```

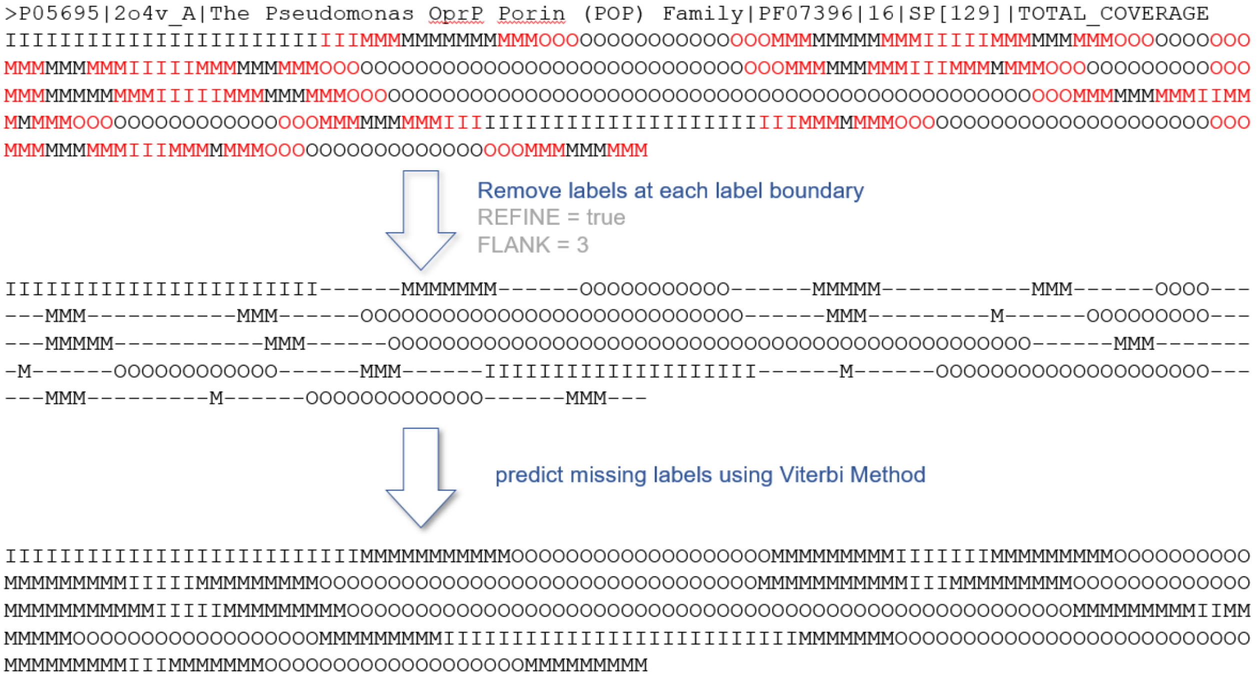
In other cases, users can write their own function to implement a different functionality.

5.7. Refine

A technique to accommodate misplaced borders in the training data is to ‘dilute’ the labels by unlabelling a few states at each label boundary (i.e. between loop and membrane regarding membrane proteins) in order to allow for some freedom in choosing the state and correcting mislabeled regions. After a model was initially estimated, the labels of the sequences were deleted in a region flanking a number of residues (i.e. in each direction of the end of a membrane-spanning segment), and predictions were performed from the relabeled sequences using the modified Viterbi algorithm presented above, and the model that was estimated from previous step. This gives a new labeling consistent with the overall structure of the protein, but with the exact boundaries moved such that it fits the model better, see Figure 2.

Configuration Settings

```
# Refine OPTIONS
FLANK=3
REFINE=true
```



```
>P05695|2o4v_A|The Pseudomonas OprP Porin (POP) Family|PF07396|16|SP[129]|TOTAL_COVERAGE
IIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMMMOOOOOOOOOOOOOOOOOMMMMMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOO
MMMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIMMMMMMMOOOOOOOOOOOOOOO
MMMMMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMIIMM
MMMMMOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOO
MMMMMMMMMIIIMMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMMM
```

Remove labels at each label boundary
REFINE = true
FLANK = 3

```
IIIIIIIIIIIIIIIIIIIIIII------MMMMMMM------OOOOOOOOOOO------MMMMM-----------MMM------OOOO---
---MMM-----------MMM------OOOOOOOOOOOOOOOOOOOOOOOOOOOO------MMM---------M------OOOOOOOOO---
---MMMMM-----------MMM------OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO------MMM-------
-M------OOOOOOOOOOOO------MMM------IIIIIIIIIIIIIIIIIIII------M------OOOOOOOOOOOOOOOOOOOO---
---MMM---------M------OOOOOOOOOOOOO------MMM---
```

predict missing labels using Viterbi Method

```
IIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIMMMMMMMMMOOOOOOOOOO
MMMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMIIIMMMMMMMMMOOOOOOOOOOOOO
MMMMMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMIIMM
MMMMMOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOO
MMMMMMMMMIIIMMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMMM
```

Fig. 2. Refine method workflow example. The lines after the sequence header show the labeling of the sequence. The second part shows the labeling after the boundaries have been unlabeled by 3 residues to each side (unlabeled positions are indicated with a “-”). Finally, a prediction is shown, which was obtained by forcing the prediction to conform to the labels.

5.8. Statistics

All the prediction algorithms also incorporate the corresponding reliability measures that have been proposed [13]. For measures of accuracy in case of membrane protein prediction methods (e.g. HMM-TM and PRED-TMBB2) we used the fraction of the correctly predicted residues in a two-state mode (Q2), the fraction of proteins with correctly predicted topologies, the segments over-lap measure (SOV) [14] and the Mathews correlation coefficient (MCC) [39] that summarizes in a single measure true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). In other cases, users can use decoding output to implement their measures.

5.9. Random Sequences Generator

Generation of random sequences according to the specified model.

```
java hmm/RandomSeq ../tables/A_DEMO ../tables/E_DEMO ../conf/conf.demo ../models/demo.mdel 10
```

Configuration Settings

```
#RANDOM SEQUENCE UTILITY
```

5.10. Multithreaded parallelization for multicores

JUCHMME supports multicore parallelization using Java Fork-Join Framework. By default, our multithreaded programs will use all available cores on your machine. The user can control the number of cores each JUCHMME process will use for computation, and disable multithreading in the configuration file. If you specify *PARALLEL=false*, the program will run in serial mode. This might be useful if you suspect something is awry with the threaded parallel implementation.

```
# Multithreaded parallelization for multicores
PARALLEL=true
allCPU=true
nCPU=2
```

5.11. Multiple sequence alignments

In many protein structure prediction problems, a significant gain in prediction accuracy can be obtained by incorporating evolutionary information in the form of multiple sequence alignments (MSAs). The user can pass a Multiple Sequence Alignment file through command line parameters.

6. Special features / Extensions

Some features listed above are uniquely supported by JUCHMME and we will discuss them briefly in the following sections.

6.1. Semi-supervised Learning

JUCHMME supports a method for semi-supervised learning of HMMs that can incorporate labeled, unlabeled and partially-labeled data in a straightforward manner [18]. The algorithm is based on a variant of the Expectation-Maximization (EM) algorithm, where the missing data are considered as the missing labels of the unlabeled or partially-labeled data. The algorithm is an instance of the so-called self-training approach, which is a commonly used technique for semi-supervised learning [40].

(1) Use the completely labeled data ($\mathbf{x}^l$, $\mathbf{y}^l$) to train an initial model (θ).

(2) Use θ to predict the labels ($\mathbf{y}^*$) of the unlabeled or partially labeled data ($\mathbf{x}^{u}$).

(3) Use the newly labeled data ($\mathbf{x}^u$, $\mathbf{y}^*$) along with the completely labeled ones ($\mathbf{x}^l$, $\mathbf{y}^l$) in order to train a new model (θ^*).

Method 1. Use all

Method 2. Use all weighted by a (constant) value of $\lambda<1$

Method 3. Use all weighted by $P(\mathbf{y}^*|\mathbf{x}^u,\theta)$

Method 4. Use most confident, i.e. those that have reliability less than an optimal threshold

(4) Remove the predicted labels ($\mathbf{y}^*$) in order to obtain the initial dataset. Use the new model θ^* to replace θ.

(5) Iterate steps (2)-(4) until convergence.

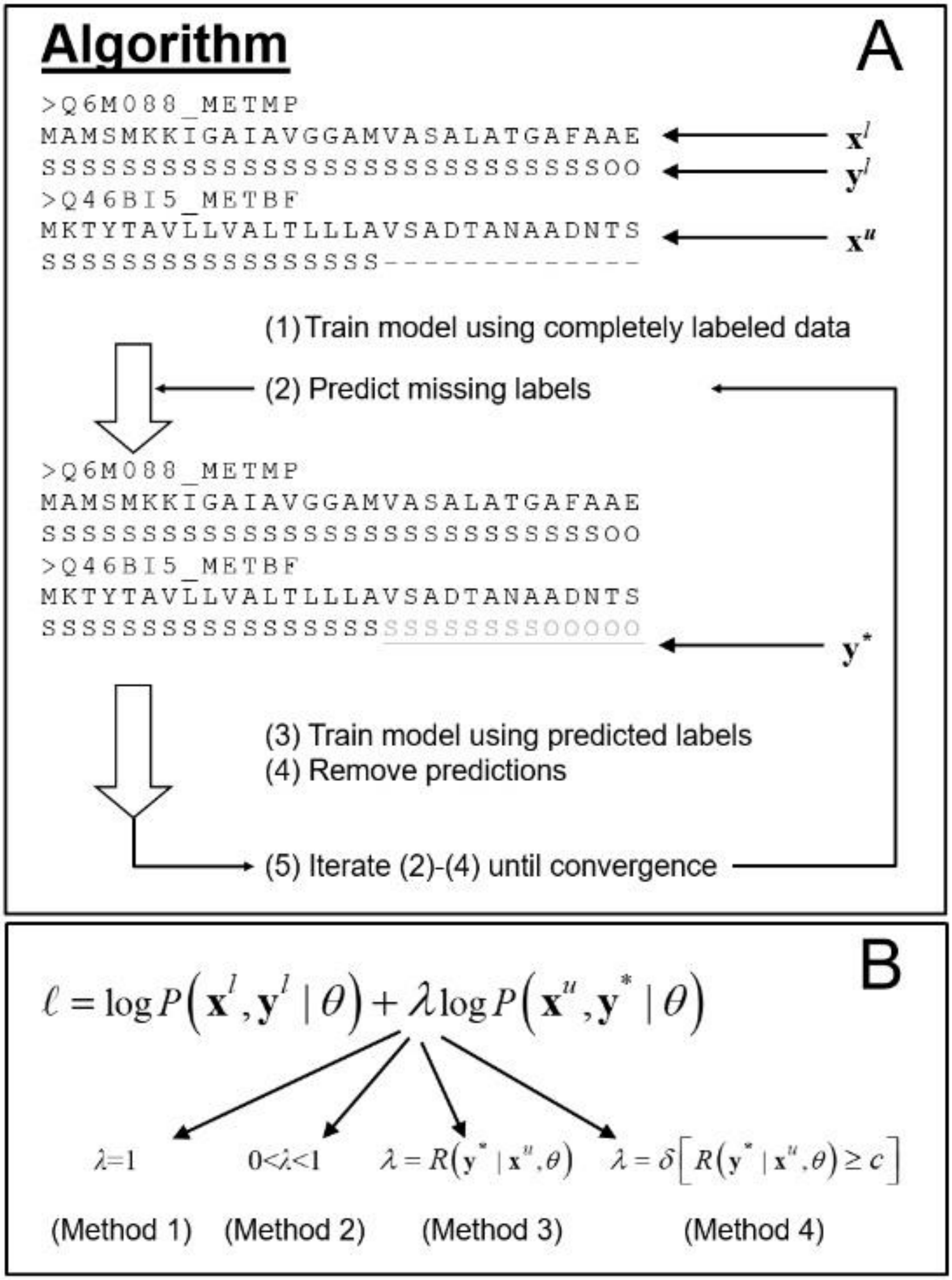

Algorithm
A
>Q6M088_METMP
MAMSMKKIGAIAVGGAMVASALATGAFAAE
SSSSSSSSSSSSSSSSSSSSSSSSSSSSOO
>Q46BI5_METBF
MKTYTAVLLVALTLLLAVSADTANAADNTS
SSSSSSSSSSSSSSSSS-------------
$\mathbf{x}^l$
$\mathbf{y}^l$
$\mathbf{x}^u$
(1) Train model using completely labeled data
(2) Predict missing labels
>Q6M088_METMP
MAMSMKKIGAIAVGGAMVASALATGAFAAE
SSSSSSSSSSSSSSSSSSSSSSSSSSSSOO
>Q46BI5_METBF
MKTYTAVLLVALTLLLAVSADTANAADNTS
SSSSSSSSSSSSSSSSSSSSSSSSSOOOOO
$\mathbf{y}^*$
(3) Train model using predicted labels
(4) Remove predictions
(5) Iterate (2)-(4) until convergence
B
$\ell = \log P\left(\mathbf{x}^l, \mathbf{y}^l \mid \theta\right) + \lambda \log P\left(\mathbf{x}^u, \mathbf{y}^* \mid \theta\right)$
$\lambda = 1$
$0 < \lambda < 1$
$\lambda = R\left(\mathbf{y}^* \mid \mathbf{x}^u, \theta\right)$
$\lambda = \delta\left[R\left(\mathbf{y}^* \mid \mathbf{x}^u, \theta\right) \geq c\right]$
(Method 1)
(Method 2)
(Method 3)
(Method 4)

Fig. 3. **A**. A schematic illustration of the algorithm. **B.** The likelihood that is being maximized in the four variants of the algorithm described in the text.

Configuration Settings

```
#SEMI-SUPERVISED LEARNING OPTIONS
SSL_ENABLED=true  (!important to enable extension)
# SSL (standard Semi-supervised Method) or GEM (Generalized EM)
SSL_METHOD=SSL
#1:  Use  all,  2:  Use  weight  (Constant)  for  each  sequence,  3:  Use  weight
(Reliability) for each sequence, 4: Use a few most confident
SSL_ADD_METHOD=1
#1:VITERBI, 2:NBEST, 3:POSVIT, 4:PLP
SSL_USING_METHOD=4
SSL_THRESHOLD=0.000002
SSL_maxIter=200
SSL_relscore = 0.95
SSL_WEIGHT=0.2
```

6.2. Extending HMMs to allow conditioning on previous observations

Instead of having the usual emission probability distribution over letters of the alphabet, we allow a state to have the emission probability conditioned on the *n* previous letters in the sequence, which corresponds to an n^{th} order Markov chain. This extension is useful for modeling coding regions. Here, we implement a simple extension of the standard HMMs in which the current observed symbol (amino acid residue) depends both on the current state and on a series of observed previous symbols [17]. The major advantage of the method is the simplicity in the implementation, which is achieved by properly transforming the observation sequence, using an extended alphabet. Notice that the state sequence is still first order, and therefore the HMM formalism is not altered significantly and all the standard algorithms remain unchanged.

To implement the new encoding, we considered capacity issues, source code compatibility, and interoperability with other systems. The method comprises of automatic conversion of initial symbols, according to the standard character encoding Unicode. We use UTF-8 that is memory efficient and used by many operating systems and programming languages. Starting from the Latin capital letter A, our method uses serially consecutive Unicode characters to create the new alphabet. Figure 4 shows the proposed method for converting the parameters, alphabet symbols and sequences based on the selected encoding. The choice of the symbols used in the new alphabet is arbitrary and of no particular importance as long as we keep track of the correspondence. Another issue that should be addressed, after the conversion of the alphabet, is the number of probabilities of the new model and the size of the respective matrices. The transition probabilities remain unaltered, while the emission probabilities have to be changed based on the new encoding. For example, a t-order series requires 20t emission probabilities per state and thus, depending on the implementation of the HMM, the user has to modify its model accordingly.

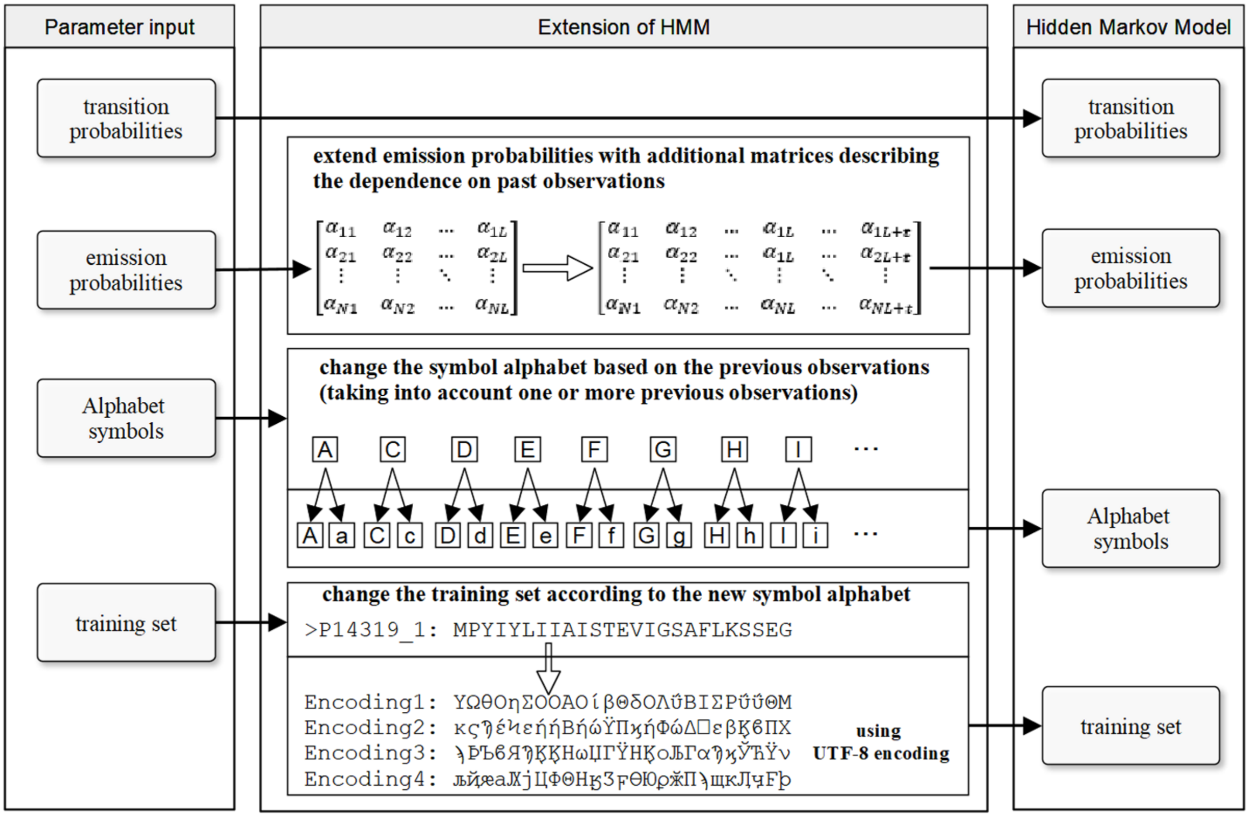

Parameter input
Extension of HMM
Hidden Markov Model
transition probabilities
emission probabilities
Alphabet symbols
training set
extend emission probabilities with additional matrices describing the dependence on past observations
$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1L} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{N1} & \alpha_{N2} & \cdots & \alpha_{NL} \end{bmatrix}$
$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1L} & \cdots & \alpha_{1L+\tau} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{1L} & \cdots & \alpha_{2L+\tau} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_{N1} & \alpha_{N2} & \cdots & \alpha_{NL} & \cdots & \alpha_{NL+\tau} \end{bmatrix}$
change the symbol alphabet based on the previous observations (taking into account one or more previous observations)
A C D E F G H I …
A a C c D d E e F f G g H h I i …
change the training set according to the new symbol alphabet
>P14319_1: MPYIYLIIAISTEVIGSAFLKSSEG
Encoding1: ΥΩθΟηΣΟΟΑΟίβΘδΟΛΰΒΙΣΡΰΰΘΜ
Encoding2: κςϡέϟεήήΒήώΫΠϰήΦώΔ□εβϏϬΠΧ
Encoding3: ϡϷЪϬЯϡϏϏΗωЏΓΫΗϏοЉΓαϡϰЎЋΫν
Encoding4: љйӕаӁјЦФѲНҕЗғѲЮϼжПϡщкЛҹҒҍ
using UTF-8 encoding
training set

Fig. 4. The HMM extension diagram.

JUCHMME supports the following encodings schemes related with protein sequences:

(1) An Encoding with 40 (20x2) symbols depending on whether the previous residue is hydrophobic (A, F, H, I, L, M, V, W, Y) or non-hydrophobic (C, D, E, G, K, N, P, Q, R, S, T). (Encoding 1).

(2) An Encoding with 80 (20x4) symbols depending on whether the previous residue is: Hydrophobic–Aromatic (F, H, Y, W), Hydrophobic–non-Aromatic (A, I, L, M, V, G), non-Hydrophobic–Charged (D, E, K, R), non-Hydrophobic–Polar (C, N, P, Q, S, T). (Encoding 2).

(3) An Encoding with 160 (20x8) symbols depending on whether the previous residue is: Hydrophobic–Small (A, G), Polar–Special (P, C), Polar–OH (S, T), Polar–NH (N, Q), Charged–Negative (D, E), Charged–Positive (K, R), Hydrophobic–Huge (I, L, M, V) and Hydrophobic–Aromatic (F, H, Y, W). (Encoding 3).

(4) An Encoding with 400 (20x20) symbols that takes into account all possible dipeptide combinations. (Encoding 4).

Configuration Settings

```
#EXTENDED PAST OBSERVATIONS
PAST_OBS_EXTENSION=true  (!important to enable extension)
#1=40, 2=80, 3=160, 4=400
ENCODE_TYPE=1
GROUP_SYMBOLS = 10
GROUPING=10001011011000000111
```

```
PAST_OBS_NO = 1
```

The user can choose one of the above encodings. In addition, JUCHMME provides the user with the ability to define their own encoding by defining the value 0 to the parameter *ENCODE_TYPE* and defining their own encoding scheme with the parameters *GROUP_SYMBOLS* and *GROUPING*.

For instance, to define an encoding with six groups, use the value 123456 which corresponds

- **1** for Group-1
- **2** for Group-2
- **3** for Group-3
- **4** for Group-4
- **5** for Group-5
- **6** for Group-6

And finally define in parameter GROUPING which group corresponds each letter of the alphabet. For instance, in the case of proteins, an observed sequence is composed by a discrete set of 20 symbols, following the alphabetical order of single-letter codes for the 20 amino acids: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y. In this case, the value will be 14431221561552443111.

Configuration Settings

```
#EXTENDED PAST OBSERVATIONS
PAST_OBS_EXTENSION=true  (!important to enable extension)
#1=40, 2=80, 3=160, 4=400
ENCODE_TYPE=0
GROUP_SYMBOLS = 123456
GROUPING=14431221561552443111
PAST_OBS_NO = 1
```

6.3. Hidden Neural Network (HNN)

Hidden Neural Network (HNN) is a general framework for hybrid Hidden Markov models (HMMs) and neural networks (NNs). The implementation of the hybrid system follows Krogh and Riis [16] framework. The basic idea in the Hidden Neural Network model is that the standard probability parameters of a CHMM are replaced by the outputs of neural networks assigned to each state. The network input w_i corresponding to x_i will usually be a window of context around x_i. Defines the left (L) and right (R) the window can be

(1) $L=R$, $L>0$, $R>0$, a symmetrical context window of $(L + R) + 1$ observations, $x_{i-L}, \ldots, x_i, \ldots, x_{i+R}$.

(2) $L \neq R$, $L>0$, $R>0$, an asymmetric context window of $(L + R) + 1$ observations, $x_{i-L}, \ldots, x_i, \ldots, x_{i+R}$.

(3) $L \neq R$, $L>0$, $R=0$, a left context window of $(L + 1)$ observations, $x_{i-L}, \ldots, x_i$.

(4) $L \neq R$, $L=0$, $R>0$, a right context window of $(R + 1)$ observations, $x_i, \ldots, x_{i+R}$.

The initialization of the weights is implemented using the JOONE (Java Object Oriented Neural Network) library (http://www.joone.org/) which is a Java framework to build and run AI applications based on neural networks.

The HNN framework uses feed-forward multilayer perceptron networks which are trained with the back-propagation algorithm. The network architecture, currently, consists of one hidden layer, one output layer and an input layer that consists of the current residue of a preset symmetrical or asymmetric window size, according to a SPARCE encoding and/or another form of a BLOSUM encoding. For the hidden layer activation function, the user can choose among Sigmoid (13), modified sigmoid (14) and hyperbolic tangent function, that limits its output within the range –1 and +1 (15). As for the output level, the activation function selected is the sigmoid function (13), since it has a range of values in the interval (0,1) and therefore fulfills the criteria so that its effect can reflect the possibility of an amino acid.

$$y = h(x) = \frac{1}{1+e^{-x}} \tag{13}$$

$$y = h(x) = \frac{1}{1+e^{-x}} - \frac{1}{2} \tag{14}$$

$$y = h(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{15}$$

Moreover, for network training it uses two different error functions, the root-mean-square error (RMSE) and Cross Entropy (CE). The neural network stops the training epochs, when it finishes a number of cycles or when the value of the global error changes during the training phase less than a predefined cutoff.

Configuration Settings

```
# TRAINING OPTIONS
RUN_CML= true  (!important to enable Conditional Maximum Likelihood Learning)
RUN_GRADIENT=true
HNN= true  (!important to enable HNN extension)

#HNN OPTIONS
windowLeft=3
windowRigth=3
nhidden=3
ADD_GRAD=0.0
DECAY=0.001
#1: Sigmoid, 2: Sigmoid Modified, 3: Tanh
hiddenLayerFunction=2

#BOOTSTRAP OPTIONS (HNN ONLY)
BOOT=0
```

```
STDEV=1.5
RANGE=5.0
SEED=568381
WEIGHT_RAND=0.0
WEIGHT_TIME=false

#RPROPNN OPTIONS (HNN ONLY)
numberOfCycles=50
doCrossVal=true
crossValIte=5
minGEdiff=0
globalError=CE
initialDelta= 0.1
maxDelta=50
minDelta=1e-6
etaInc=1.2
etaDec=0.5
```

7. Examples

JUCHMMER has already been used in a number of important biological problems. The JUCHMME software package comprises several examples with all the necessary input and output files. The following examples illustrate the range of features supported by JUCHMME.

7.1. PRED-TMMB

This HMM can be used to predict the topology of transmembrane β-barrels [27, 41].

This example shows how parameter training with JUCHMME works.

<u>Self-consistency</u>

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TMBB2 -e ../tables/E_TMBB2 -c
../conf/conf.tmbb -m ../models/tmbb.mdel -t ../input/TRAIN_SET_49_ALL.for_train -s
```

<u>Jackknife</u>

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TMBB2 -e ../tables/E_TMBB2 -c
../conf/conf.tmbb -m ../models/tmbb.mdel -t ../input/TRAIN_SET_49_ALL.for_train -j
```

<u>k-fold Cross-validation (k=10)</u>

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TMBB2 -e ../tables/E_TMBB2 -c
../conf/conf.tmbb -m ../models/tmbb.mdel -t ../input/TRAIN_SET_49_ALL.for_train -v 10
```

HNN Self-consistency (RPROP Method for initialize Weights)

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TMBB2 -e ../tables/E_TMBB2 -w
../tables/W_PREDTMBB2_MYMODEL -c ../conf/conf.tmbb -m ../models/tmbb.mdel -x
../tables/SPARCE -t ../input/TRAIN_SET_49_ALL.for_train -s
```

Training

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TMBB2 -e ../tables/E_TMBB2 -c
../conf/conf.tmbb -m ../models/tmbb.mdel -t ../input/TRAIN_SET_49_ALL.for_train
```

Testing

```
java -Xmx1024m hmm/Juchmme -a ../tables/A_TMBB2_TRAINED -e
../tables/E_TMBB2_TRAINED -c ../conf/conf.tmbb -m ../models/tmbb.mdel -f ../input/
TRAIN_SET_49_ALL.fasta
```

Testing using previous knowledge

```
java -Xmx1024m -Dfile.encoding=UTF-8 hmm/Juchmme -a ../tables/A_TMBB2_TRAINED -e
../tables/E_TMBB2_40_TRAINED -c ../conf/conf.tmbb -m ../models/tmbb.mdel -f ../input/PRED-
TMBB2_newSet_40.seq
```

7.2. HMM-TM

This HMM can be used to predict α-helical transmembrane proteins [11].

This example shows how parameter training with JUCHMME works.

Self-consistency

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_HELICAL -e ../tables/E_HELICAL -c
../conf/conf.hmmtm -m ../models/hmmtm.mdel -t ../input/hmmtm_train_set.3line -s
```

Jackknife

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_HELICAL -e ../tables/E_HELICAL -c
../conf/conf.hmmtm -m ../models/hmmtm.mdel -t ../input/ hmmtm_train_set.3line -j
```

Training

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_HELICAL -e ../tables/E_HELICAL -c
../conf/conf.hmmtm -m ../models/hmmtm.mdel -t ../input/ hmmtm_train_set.3line
```

Testing

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_HELICAL_TRAINED -e
../tables/E_HELICAL_TRAINED -c ../conf/conf.hmmtm -m ../models/hmmtm.mdel -f
../input/hmmtm_test_set.fasta
```

Testing using previous knowledge

```
java -Xmx1024m -Dfile.encoding=UTF-8 hmm/Juchmme -a ../tables/A_HELICAL_TRAINED -e
../tables/E_HELICAL_40_TRAINED -c ../conf/conf. hmmtm -m ../models/hmmtm.mdel -f
../input/hmmtm_train_set.3line
```

7.3. PRED-TAT

This HMM can be used to predict twin-arginine and secretory signal peptides [42].

This example shows how parameter training with JUCHMME works.

Self-consistency

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TAT -e ../tables/E_TAT -c ../conf/conf.tat -m
models/tat.mdel -t ../input/TAT_Train_Set.crossval -s
```

k-fold Cross-validation (k=31)

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TAT -e ../tables/E_TAT -c ../conf/conf.tat -m
../models/tat.mdel -t ../input/TAT_Train_Set.crossval -v 31
```

Training

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_TAT_TRAINED -e ../tables/E_TAT_TRAINED -c
../conf/conf.tat -m models/tat.mdel -t ../input/TAT_Test_Set.fasta
```

Testing

```
java -Xmx1024m hmm/juchmme -a ../tables/ A_TAT_TRAINED -e ../tables/ E_TAT_TRAINED -c
../conf/conf.tat  -m ../models/tat.mdel -f ../input/ TAT_Train_Set_ALL.fasta
```

Testing using previous knowledge

```
java -Xmx1024m -Dfile.encoding=UTF-8 hmm/Juchmme -a ../tables/ A_TAT_TRAINED -e
../tables/ E_TAT_40_TRAINED -c ../conf/conf.tat  -m ../models/tat.mdel -f ../input/
TAT_Test_Set.fasta
```

7.4. PRED-LIPO

This HMM can be used to predict lipoprotein signal peptides in Gram-positive bacteria [23].

This example shows how parameter training with JUCHMME works.

Self-consistency

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_LIPO -e ../tables/E_LIPO -c ../conf/conf.lipo -m
../models/lipo.mdel -t ../input/LIPO_Train_Set.seq -s
```

k-fold Cross-validation (k=11)

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_LIPO -e ../tables/E_LIPO -c ../conf/conf.lipo -m
../models/lipo.mdel -t ../input/LIPO_Train_Set.seq -v 11
```

Training

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_LIPO_TRAINED -e ../tables/E_LIPO_TRAINED
-c ../conf/conf.lipo -m ../models/lipo.mdel -f ../input/LIPO_Test_Set.fasta
```

7.5. PRED-SIGNAL

This HMM can be used to predict signal peptides in archea [22].

This example shows how parameter training with JUCHMME works.

Self-consistency

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_ARCHAEA -e ../tables/E_ARCHAEA -c
../conf/conf.signal -m ../models/signal.mdel -t ../input/ARCHAEA_Train_Set.seq -s
```

k-fold Cross-validation (k=9)

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_ARCHAEA -e ../tables/E_ARCHAEA -c
../conf/conf.signal -m ../models/signal.mdel -t ../input/Signal-ver+ref_shuffled68.seq -v 9
```

Training

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_ARCHAEA -e ../tables/E_ARCHAEA -c
../conf/conf.signal -m ../models/signal.mdel -t ../input/Signal-ver+ref_shuffled68.seq
```

7.6. LPXTG

This HMM can be used to predict the LPXTG and LPXTG-like cell-wall proteins of Gram-positive bacteria [43].

Testing

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_LPX -e ../tables/E_LPX -c ../conf/conf.tmbb -m
../models/lpxtg.mdel -f ../input/LPXTG_Test_Set.fasta
```

7.7. HMMpTM

This HMM can be used to predict the topology of transmembrane proteins and the existence of kinase specific phosphorylation and N/O-linked glycosylation sites along the protein sequence [44].

Testing

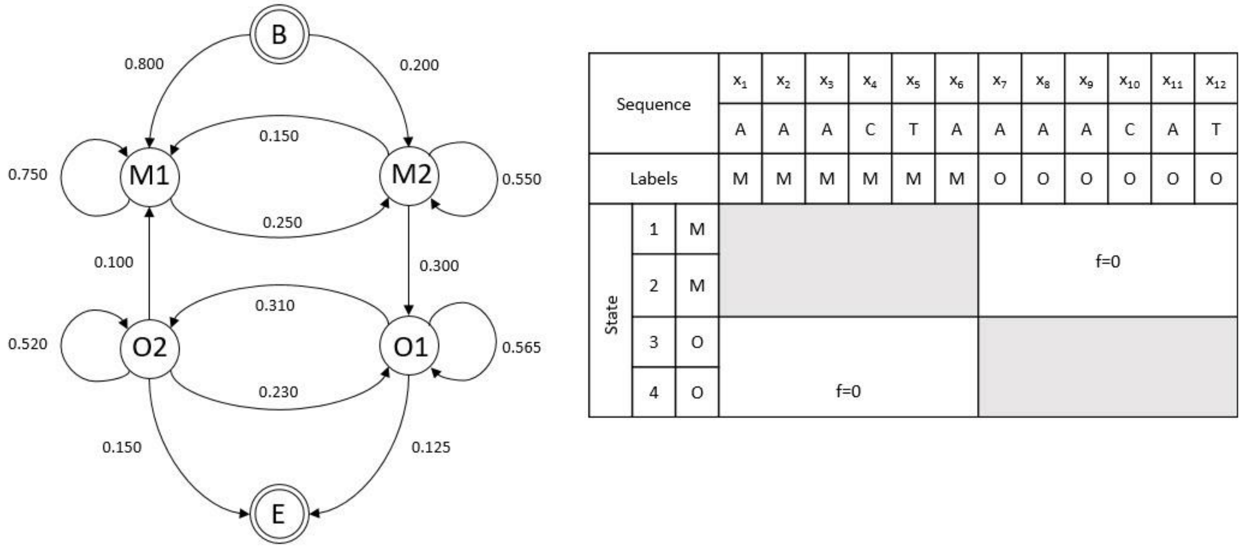
```
java -Xmx4096m hmm/Juchmme -a ../tables/A_HELICAL_PTM_v6 -e
../tables/E_HELICAL_PTM_v6 -c ../conf/conf.hmmptm -m ../models/hmmptm.mdel -f
../input/hmmptm_test_set.fasta
```

Testing using previous knowledge

```
java -Xmx1024m -Dfile.encoding=UTF-8 hmm/Juchmme -a ../tables//A_HELICAL_PTM_v6 -e
../tables/ E_HELICAL_PTM_v6 -c ../conf/conf. hmmptm -m ../models/hmmptm.mdel -f
../input/hmmptm_train_set.3line
```

7.8. DEMO

A demo model which describes the so-called Class HMM (CHMM)



Sequence			x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
			A	A	A	C	T	A	A	A	A	C	A	T
Labels			M	M	M	M	M	M	O	O	O	O	O	O
State	1	M							f=0					
	2	M												
	3	O	f=0											
	4	O												

Fig. 5. Left. A schematic illustration of a very simple model with four states, two with label (M) and two with label (O). The model includes also begin (B) and end (E) states. The model uses multiple label states. **Right.** An example observation sequence **x**=x_1, x_2, …,x_{12} with complete labels and the multiple label states. The grey areas of the matrix are calculated as in the standard HMM algorithms whereas f is set to zero in the white areas.

Random Sequences Creator

```
java hmm/RandomSeq ../tables/A_DEMO ../tables/E_DEMO ../conf/conf.demo
../models/demo.mdel 100 > demoSet.seq
```

Self-consistency

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_DEMO -e ../tables/E_DEMO -c
../conf/conf.demo -m ../models/demo.mdel -t ../input/demoSet.seq -s
```

k-fold Cross-validation (k=9)

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_DEMO -e ../tables/E_DEMO -c
../conf/conf.demo -m ../models/demo.mdel -t ../input/ demoSet.seq -v 9
```

Training

```
java -Xmx4096m hmm/Juchmme -a ../tables/A_DEMO -e ../tables/E_DEMO -c
../conf/conf.demo -m ../models/demo.mdel -t ../input/demoSet.seq
```

8. Future developments

Training, decoding and applying HMMs are subjects of active research in our lab, and thus JUCHMME will be continuously updated.

Web edition: Additionally, we are planning to construct a simple and user-friendly web interface that will allow the design of input parameter files in a graphical environment. A graphical illustration of the model will help people towards better understanding of the biological problem and HMM structure of the model, and will be especially useful for educational purposes.

New algorithms: We are planning on implementing new and advanced algorithms, like linear memory EM and Viterbi algorithms [45], linear-memory Baum-Welch training [46], efficient algorithms for training the parameters of Hidden Markov Models using stochastic expectation maximization (EM) training and Viterbi training [47]. Finally, we are working with other types of extensions to the standard HMM architecture, either by allowing silent states or by introducing novel hybrid HMM methods.

References

1. Baum, L.E., *An equality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes.* Inequalities, 1972. **3**: p. 1-8.
2. Durbin, R., et al., *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. 1998: Cambridge university press.
3. Rabiner, L.R., *A tutorial on hidden Markov models and selected applications in speech recognition.* Proceedings of the IEEE, 1989. **77**(2): p. 257-286.
4. Dempster, A.P., N.M. Laird, and D.B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm.* Journal of the royal statistical society. Series B (methodological), 1977: p. 1-38.
5. Baldi, P. and Y. Chauvin, *Smooth on-line learning algorithms for hidden Markov models.* Neural Computation, 1994. **6**(2): p. 307-318.
6. Juang, B.-H. and L.R. Rabiner, *The segmental K-means algorithm for estimating parameters of hidden Markov models.* IEEE Transactions on acoustics, speech, and signal Processing, 1990. **38**(9): p. 1639-1641.
7. Bagos, P.G., T.D. Liakopoulos, and S.J. Hamodrakas. *Faster gradient descent training of hidden Markov models, using individual learning rate adaptation*. in *International Colloquium on Grammatical Inference*. 2004. Springer.
8. Krogh, A., *Two methods for improving performance of an HMM and their application for gene finding.* Center for Biological Sequence Analysis. Phone, 1997. **45**: p. 4525.
9. Fariselli, P., P.L. Martelli, and R. Casadio, *A new decoding algorithm for hidden Markov models improves the prediction of the topology of all-beta membrane proteins.* BMC bioinformatics, 2005. **6**(4): p. S12.
10. Käll, L., A. Krogh, and E.L. Sonnhammer, *An HMM posterior decoder for sequence feature prediction that includes homology information.* Bioinformatics, 2005. **21**(suppl_1): p. i251-i257.
11. Bagos, P.G., T.D. Liakopoulos, and S.J. Hamodrakas, *Algorithms for incorporating prior topological information in HMMs: application to transmembrane proteins.* BMC bioinformatics, 2006. **7**(1): p. 189.
12. Melen, K., A. Krogh, and G. von Heijne, *Reliability measures for membrane protein topology prediction algorithms.* Journal of molecular biology, 2003. **327**(3): p. 735-744.
13. Baldi, P., et al., *Assessing the accuracy of prediction algorithms for classification: an overview.* Bioinformatics, 2000. **16**(5): p. 412-424.
14. Zemla, A., et al., *A modified definition of Sov, a segment‐based measure for protein secondary structure prediction assessment.* Proteins: Structure, Function, and Bioinformatics, 1999. **34**(2): p. 220-223.
15. Theodoropoulou, M.C., I. Mintsopoulos, and P.G. Bagos, *Viterbi training of Hidden Markov Models for labeled sequences*, in *Joint 25th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB) and 16th European Conference on Computational Biology (ECCB)*. 2017.
16. Krogh, A. and S.K. Riis, *Hidden neural networks.* Neural Computation, 1999. **11**(2): p. 541-563.
17. Tamposis, I.A., et al., *Extending Hidden Markov Models to Allow Conditioning on Previous Observations.* Journal of Bioinformatics and Computational Biology, 2018.
18. Tamposis, I.A., et al., *Semi-supervised learning of Hidden Markov Models for biological sequence analysis.* Bioinformatics, 2018: p. bty910-bty910.
19. Krogh, A. *Hidden Markov models for labeled sequences*. in *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision &amp; Image Processing., Proceedings of the 12th IAPR International. Conference on*. 1994. IEEE.

20. Böer, J., *Multiple alignment using hidden Markov models.* proteins. **4**: p. 14.
21. Nielsen, H. and A.S. Krogh. *Prediction of signal peptides and signal anchors by a hidden Markovmodel*. in *Sixth International Conference on Intelligent Systems for Molecular Biology*. 1998. AAAI Press.
22. Bagos, P., et al., *Prediction of signal peptides in archaea.* Protein Engineering Design and Selection, 2009. **22**(1): p. 27-35.
23. Bagos, P.G., et al., *Prediction of lipoprotein signal peptides in Gram-positive bacteria with a Hidden Markov Model.* Journal of proteome research, 2008. **7**(12): p. 5082-5093.
24. Juncker, A.S., et al., *Prediction of lipoprotein signal peptides in Gram‐negative bacteria.* Protein Science, 2003. **12**(8): p. 1652-1662.
25. Litou, Z.I., et al., *Prediction of cell wall sorting signals in gram-positive bacteria with a hidden markov model: application to complete genomes.* Journal of bioinformatics and computational biology, 2008. **6**(02): p. 387-401.
26. Asai, K., S. Hayamizu, and K.i. Handa, *Prediction of protein secondary structure by the hidden Markov model.* Bioinformatics, 1993. **9**(2): p. 141-146.
27. Bagos, P.G., et al., *A Hidden Markov Model method, capable of predicting and discriminating β-barrel outer membrane proteins.* BMC bioinformatics, 2004. **5**(1): p. 29.
28. Krogh, A., et al., *Predicting transmembrane protein topology with a hidden markov model: application to complete genomes1.* Journal of molecular biology, 2001. **305**(3): p. 567-580.
29. Bagos, P.G., T.D. Liakopoulos, and S.J. Hamodrakas, *Evaluation of methods for predicting the topology of β-barrel outer membrane proteins and a consensus prediction method.* BMC bioinformatics, 2005. **6**(1): p. 7.
30. Möller, S., M.D. Croning, and R. Apweiler, *Evaluation of methods for the prediction of membrane spanning regions.* Bioinformatics, 2001. **17**(7): p. 646-653.
31. Viklund, H. and A. Elofsson, *Best α‐helical transmembrane protein topology predictions are achieved using hidden Markov models and evolutionary information.* Protein Science, 2004. **13**(7): p. 1908-1917.
32. Jones, D.T., W.R. Taylor, and J.M. Thornton, *A model recognition approach to the prediction of all-helical membrane protein structure and topology.* Biochemistry, 1994. **33**(10): p. 3038-49.
33. Fariselli, P., et al., *MaxSubSeq: an algorithm for segment-length optimization. The case study of the transmembrane spanning segments.* Bioinformatics, 2003. **19**(4): p. 500-5.
34. Jones, D., W. Taylor, and J. Thornton, *A model recognition approach to the prediction of all-helical membrane protein structure and topology.* Biochemistry, 1994. **33**(10): p. 3038-3049.
35. Martelli, P.L., et al., *A sequence-profile-based HMM for predicting and discriminating β barrel membrane proteins.* Bioinformatics, 2002. **18**(suppl_1): p. S46-S53.
36. Riedmiller, M. and H. Braun. *RPROP-A fast adaptive learning algorithm*. in *Proc. of ISCIS VII), Universitat*. 1992. Citeseer.
37. Krogh, A. and J.A. Hertz. *A simple weight decay can improve generalization*. in *Advances in neural information processing systems*. 1992.
38. Schwartz, R. and Y.-L. Chow. *The N-best algorithm: An efficient and exact procedure for finding the N most likely sentence hypotheses*. in *Proc. ICASSP*. 1990.
39. Matthews, B.W., *Comparison of the predicted and observed secondary structure of T4 phage lysozyme.* Biochimica et Biophysica Acta (BBA)-Protein Structure, 1975. **405**(2): p. 442-451.
40. Yarowsky, D. *Unsupervised word sense disambiguation rivaling supervised methods*. in *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. 1995. Association for Computational Linguistics.

41. Tsirigos, K.D., A. Elofsson, and P.G. Bagos, *PRED-TMBB2: improved topology prediction and detection of beta-barrel outer membrane proteins.* Bioinformatics, 2016. **32**(17): p. i665-i671.
42. Bagos, P.G., et al., *Combined prediction of Tat and Sec signal peptides with hidden Markov models.* Bioinformatics, 2010. **26**(22): p. 2811-2817.
43. Fimereli, D.K., et al. *CW-PRED: a HMM-based method for the classification of cell wall-anchored proteins of Gram-positive bacteria*. in *Hellenic Conference on Artificial Intelligence*. 2012. Springer.
44. Tsaousis, G.N., P.G. Bagos, and S.J. Hamodrakas, *HMMpTM: improving transmembrane protein topology prediction using phosphorylation and glycosylation site prediction.* Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics, 2014. **1844**(2): p. 316-322.
45. Churbanov, A. and S. Winters-Hilt, *Implementing EM and Viterbi algorithms for Hidden Markov Model in linear memory.* BMC bioinformatics, 2008. **9**(1): p. 224.
46. Miklós, I. and I.M. Meyer, *A linear memory algorithm for Baum-Welch training.* BMC bioinformatics, 2005. **6**(1): p. 231.
47. Lam, T.Y. and I.M. Meyer, *Efficient algorithms for training the parameters of hidden Markov models using stochastic expectation maximization (EM) training and Viterbi training.* Algorithms for Molecular Biology, 2010. **5**(1): p. 38.