Conformal Prediction For Protein Sequences

Sathwik Pajjuri

Submitted for the Degree of Master of Science in

Data Science And
Analytics

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

November 26th , 2021

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 6,000 words.
Student Name: Sathwik Pajjuri
Date of Submission: 1 December 2021
Signature:

Acknowledgements

I would like to express my gratitude to my supervisor Dr Shanahan Hugh who has shaped, guided and refined my work through this experiment. His subject expertise, his intuition on the areas to explore and his patience as a teacher played a major part in making this project what it is today

Abstract

The Objective of this project is to run Conformal Prediction Algorithm on various Protein Sequences available in the internet. Estimating the reliability of individual predictions is key to increase the adoption of computational models and 'artificial intelligence' in predicting protein sequences, as well as to foster its application to guide decision making in clinical trails. Among the large number of algorithms developed over the last decades to compute prediction errors, Conformal Prediction (CP) has gained increasing attention in the analyzing protein sequences. A major reason for its recent popularity is the ease of interpretation of the computed prediction errors in both classification and regression tasks. For instance, at a confidence level of 90% the true value will be within the predicted confidence intervals in at least 90% of the cases. This so called validity of conformal predictors is guaranteed by the robust mathematical foundation underlying CP. The versatility of CP relies on its minimal computational footprint, as it can be easily coupled to any machine learning algorithm at little computational cost. In this review, we summarize underlying concepts and practical applications of CP with a particular focus on Multiple Sequence Alignments and Hidden Markov Models, and list open source implementations of relevant software. Finally, we describe the current limitations in the field, and provide a perspective on future opportunities for CP in analyzing protein sequences.

Contents

1 Introduction

Conformal Prediction uses past experience to determine the precise level of confidence in new predictions. Given an error probability ε, together with a method that makes a prediction ŷ of a label y, it produces a set of labels, typically containing ŷ, that also contains y with probability 1 − ε. In decision-making, Artificial Intelligence (AI) systems need to not only make predictions but also quantify their predictions' certainty (uncertainty). For example, consider a stock automatic trading system where an AI system predicts the stock price. A point prediction from the AI system might be dramatically different from the real value because of the high stochasticity of the stock market. But, on the other hand, if the AI system could estimate the range which guarantees to cover the true value with high probability, the trading system could compute the best and worst rewards and make more sensible decisions.

Conformal prediction is a technique for quantifying such uncertainties for AI systems. In particular, given an input, conformal prediction estimates a prediction interval in regression problems and a set of classes in classification problems. Both the prediction interval and sets are guaranteed to cover the true value with high probability. A major research area in machine learning is the development of algorithms to compute the reliability of individual predictions. Such reliability estimates are essential to increase the trust and application of artificial intelligence solutions to guide decision making, especially in the context of healthcare. Estimating the reliability of predictive models is of particular relevance in predicting protein sequences, where both the prediction and the associated uncertainty need to be taken into account for decision making. The set of molecules for which a model is expected to generate reliable predictions is termed the applicability domain of a model. Therefore, the development of algorithms to define and compute the applicability domain of predictive models has been an area of intense research in applying machine learning models to bioinformatics, and due to the amount of recent research in the field we will in the following often focus on Multiple Sequence Alignment(MSA's) and Hidden Markov Models(HMM's). Multiple sequence alignment is often used to assess sequence conservation of protein domains, tertiary and secondary structures, and even individual amino acids or nucleotides. Where as Hidden Markov models in biological sequence analysis. used to solve various sequence analysis problems, such as pairwise and multiple sequence alignments, gene annotation, classification, similarity search, and many others.[2][1]

2 Background Research

2.1 PF00069_seed Dataset:

The protein kinase domain is a structurally conserved protein domain containing the catalytic function of protein kinases. Protein kinases are a group of enzymes that move a phosphate group onto proteins, in a process called phosphorylation. This functions as an on/off switch for many cellular processes, including metabolism, transcription, cell cycle progression, cytoskeletal rearrangement and cell movement, apoptosis, and differentiation. They also function in embryonic development, physiological responses, and in the nervous and immune system.

The general purpose of the Pfam database is to provide a complete and accurate classification of protein families and domains. Originally, the rationale behind creating the database was to have a semi-automated method of curating information on known protein families to improve the efficiency of annotating genomes. The Pfam classification of protein families has been widely adopted by biologists because of its wide coverage of proteins and sensible naming conventions.

It is used by experimental biologists researching specific proteins, by structural biologists to identify new targets for structure determination, by computational biologists to organise sequences and by evolutionary biologists tracing the origins of proteins. Early genome projects, such as human and fly used Pfam extensively for functional annotation of genomic data. The Pfam website allows users to submit protein or DNA sequences to search for matches to families in the database. If protein is submitted, a six-frame translation is performed, then each frame is searched. Rather than performing a typical BLAST search, Pfam uses profile hidden Markov models, which give greater weight to matches at conserved sites, allowing better remote homology detection, making them more suitable for annotating genomes of organisms with no well-annotated close relatives.

2.2 Multiple Sequence Alignments

Multiple sequence alignment (MSA) may refer to the process or the result of sequence alignment of three or more biological sequences, generally protein, DNA, or RNA. In many cases, the input set of query sequences are assumed to have an evolutionary relationship by which they share a linkage and are descended from a common ancestor. From the resulting MSA, sequence homology can be inferred and phylogenetic analysis can be conducted to assess the sequences' shared evolutionary origins. Visual depictions of the alignment as in the image at right illustrate mutation events such as point mutations (single amino acid or nucleotide changes) that appear as differing characters in a single alignment column, and insertion or deletion mutations (indels or gaps) that appear as hyphens in one or more of the sequences in the alignment.

Multiple sequence alignment is often used to assess sequence conservation of protein domains, tertiary and secondary structures, and even individual amino acids or nucleotides.

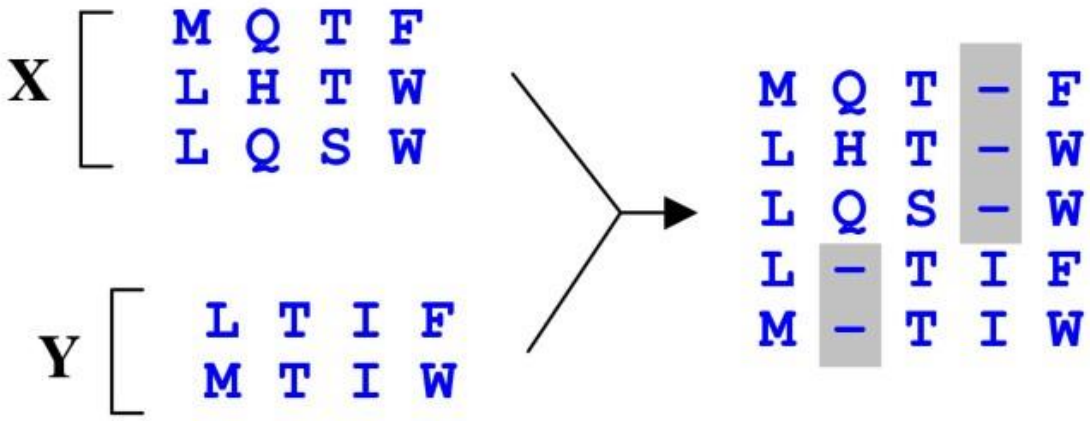
Given m Sequences, S_i , i=1,…..,m

$$S := \begin{cases} S_1 = (S_{11}, S_{12}, \ldots, S_{1n_1}) \\ S_2 = (S_{21}, S_{22}, \cdots, S_{2n_2}) \\ \quad \vdots \\ S_m = (S_{m1}, S_{m2}, \ldots, S_{mn_m}) \end{cases}$$

A multiple sequence alignment is taken of this set of sequences S' by inserting by inserting amounts of gaps needed into each of the S_i sequences of S until the modified sequences, S_i' all conform to length L>max{ n_i | i=1,…,m} and no values in the sequences of S of the same column consists of only gaps. The mathematical form of an MSA of the above sequence set is shown below:

$$S' := \begin{cases} S'_1 = (S'_{11}, S'_{12}, \ldots, S'_{1L}) \\ S'_2 = (S'_{21}, S'_{22}, \ldots, S'_{2L}) \\ \quad \vdots \\ S'_m = (S'_{m1}, S'_{m2}, \ldots, S'_{mL}) \end{cases}$$

To return from each particular sequence S_i' to S_i ,remove all gaps.[2][1]



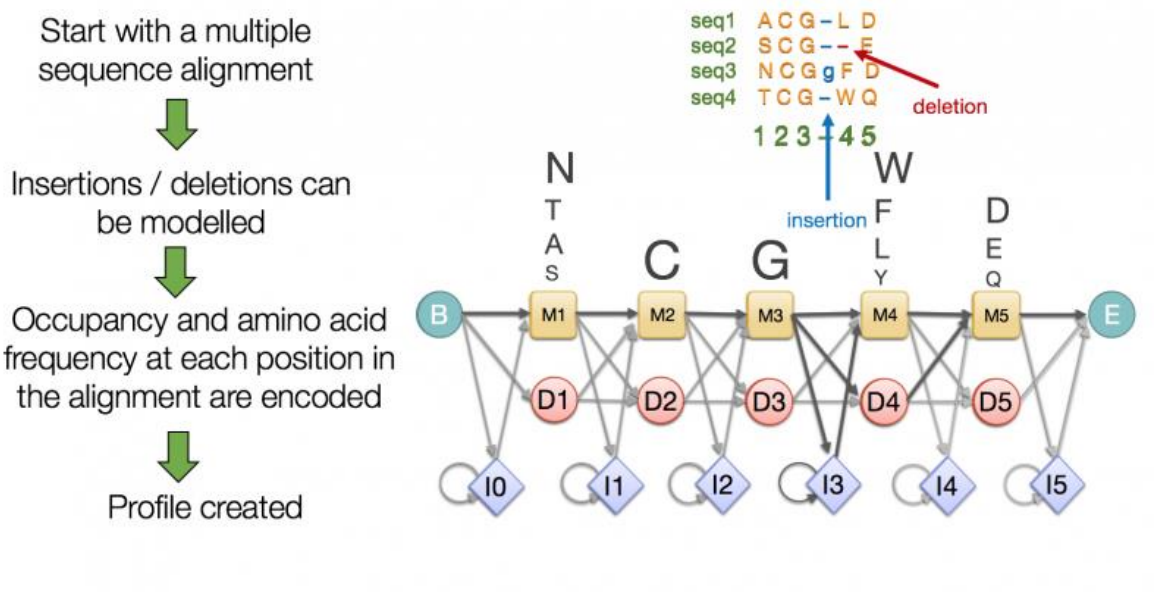
2.3 Hidden Markov Models

Hidden Markov models (HMMs), named after the Russian mathematician Andrey Andreyevich Markov, who developed much of relevant statistical theory, are introduced and studied in the early 1970s. They were first used in speech recognition and have been successfully applied to the analysis of biological sequences since late 1980s. Nowadays, they are considered as a specific form of dynamic Bayesian networks, which are based on

the theory of Bayes. HMMs are statistical models to capture hidden information from observable sequential symbols (e.g., a nucleotidic sequence).

They have many applications in sequence analysis, in particular to predict exons and introns in genomic DNA, identify functional motifs (domains) in proteins (profile HMM), align two sequences (pair HMM). In a HMM, the system being modelled is assumed to be a Markov process with unknown parameters, and the challenge is to determine the hidden parameters from the observable parameters. A good HMM accurately models the real world source of the observed real data and has the ability to simulate the source. A lot of Machine Learning techniques are based on HMMs have been successfully applied to problems including speech recognition, optical character recognition, computational biology and they have become a fundamental tool in bioinformatics: for their robust statistical foundation, conceptual simplicity and malleability, they are adapted fit diverse classification problems. In Computational Biology, a hidden Markov model (HMM) is a statistical approach that is frequently used for modelling biological sequences. In applying it, a sequence is modelled as an output of a discrete stochastic process, which progresses through a series of states that are 'hidden' from the observer. Each such hidden state emits a symbol representing an elementary unit of the modelled data, for example, in case of a protein sequence, an amino acid. In the following sections, we first introduce the concepts of Hidden Markov Model as a particular type of probabilistic model in a Bayesian framework; then, we describe some important aspects of modelling Hidden Markov Models in order to solve real problems, giving particular emphasis in its use in biological context. Hidden Markov models are probabilistic models that can assign likelihoods to all possible combinations of gaps, matches, and mismatches to determine the most likely MSA or set of possible MSAs. HMMs can produce a single highest-scoring output but can also generate a family of possible alignments that can then be evaluated for biological significance. To show the potentiality of these statistical approaches, we present the stochastic modelling of an HMM, defining first the model architecture and then the learning and operating algorithms. HMMs can produce both global and local alignments.[2][1]

Typical HMM-based methods work by representing an MSA as a form of directed acyclic graph known as a partial-order graph, which consists of a series of nodes representing possible entries in the columns of an MSA. In this representation a column that is absolutely conserved (that is, that all the sequences in the MSA share a particular character at a particular position) is coded as a single node with as many outgoing connections as there are possible characters in the next column of the alignment. In the terms of a typical hidden Markov model, the observed states are the individual alignment columns and the "hidden" states represent the presumed ancestral sequence from which the sequences in the query set are hypothesized to have descended. An efficient search variant of the dynamic programming method, known as the Viterbi algorithm, is generally used to successively align the growing MSA to the next sequence in the query set to produce a new MSA.[2][3]



A sample figure on how HMM works on Multiple Sequence Alignment.[2][1]

2.4 Conformal Prediction

Conformal prediction is a technique used for quantifying uncertainties for AI systems. In particular, given an input, conformal prediction estimates a prediction interval in regression problems and a set of classes in classification problems. Both the prediction interval and sets are guaranteed to cover the true value with high probability.

Conformal prediction can be used with any method of point prediction for classification or regression, including support-vector machines, decision trees, boosting, neural networks, and Bayesian prediction. Starting from the method for point prediction, we construct a nonconformity measure, which measures how unusual an example looks relative to previous examples, and the conformal algorithm turns this nonconformity measure into prediction regions.

In A Tutorial on Conformal Prediction, Glenn Shafer and Vladimir Vovk provide the following definition:

" Conformal prediction uses past experience to determine precise levels of confidence in new predictions."

Consider an Identically Independently Distributed (IID) datasets,

$\{(X_1, Y_1),\ldots\ldots,(X_n, Y_n)\}$,

where X's are input features and Y's are labels, n is the number of data points. We also assume that a machine learning model f: X->Y has been trained.

2.4.1 Transductive Conformal Prediction

Let $z_1=(x_1,y_1),z_2=(x_2,y_2),\ldots,z_n=(x_n,y_n)$ be successive pairs constituting the examples, with $x_i \in X$ an object and $y_i \in Y$ its label. For any sequence $z_1,z_2,\ldots,z_n \in Z^*$ and any new object $x_{n+1} \in X$, we can define a simple predictor D such as:

$D:Z^* \times X \rightarrow Y.$ ……(1)

This simple predictor D produces a point prediction $D(z_1,\ldots,z_n,x_{n+1}) \in Y$, which is the prediction for y_{n+1}, the true label of x_{n+1}.

By adding another parameter $\epsilon \in (0,1)$ which is the probability of error called the significance level, this simple predictor becomes a confidence predictor Γ that can predict a subset of Y with a confidence level $1-\epsilon$, which corresponds to a statistical guarantee of coverage of the true label y_{n+1}. Γ is defined as follows:

$\Gamma:Z^* \times X \times (0,1) \rightarrow 2^Y,$ ……(2)

where 2^Y denotes the power set of Y. This confidence predictor $\Gamma\epsilon$ must be decreasing for the inclusion with respect to ϵ, i.e. we must have:

$\forall\, n>0, \forall\, \epsilon_1 \geq \epsilon_2, \Gamma^{\epsilon 1}(z_1,\ldots,z_n,x_{n+1}) \subseteq \Gamma^{\epsilon 2}(z_1,\ldots,z_n,x_{n+1}).$ ……(3)

The two main properties desired in confidence predictors are (a) validity, meaning the error rate does not exceed ϵ for each chosen confidence level ϵ, and (b) efficiency, i.e. prediction sets are as small as possible. Therefore, a prediction set with fewer labels will be much more informative and useful than a bigger prediction set.

To build such a predictor, conformal prediction relies on a non-conformity measure A_n. This measure calculates a score that estimates how strange an example z_i is from a bag of other examples Open image in new window . We then note αi the non-conformity score of z_i compared to the other examples, such as:

$$\alpha_i := A_n\left(\left\{z_1,\ldots,z_{i-1},z_{i+1},\ldots,z_n\right\},z_i\right).$$ ……(4)

Comparing α_i with other non-conformity scores α_j with $j \neq i$, we calculate a p-value of z_i expressing the proportion of less conforming examples than z_i, with:

$|\{j=1,\ldots,n:\alpha_j \geq \alpha_i\}|/n.$ ……(5)

If the p-value approaches the lower bound 1/n then z_i is non-compliant to most other examples (an outlier). If, on the contrary, it approaches the upper bound 1 then z_i is very consistent.

We can then compute the p-value for the new example x_{n+1} being classified as each possible label $y \in Y$ by using (5). More precisely, we can consider for each $y \in Y$ the sequence $(z_1, \ldots, z_n, z_{n+1} = (x_{n+1}, y))$ and derive from that scores $\alpha^y_1, \ldots, \alpha^y_{n+1}$. We thus get a conformal predictor by predicting the set:

$\Gamma^\epsilon(x_{n+1}) = \{y \in Y : |\{i = 1, \ldots, n, n+1 : \alpha^y_i \geq \alpha^y_{n+1}\}| n+1 > \epsilon\}$. ……..(6)

Constructing a conformal predictor therefore amounts to defining a non-conformity measure that can be built based on any machine learning algorithm called the underlying algorithm of the conformal prediction. Popular underlying algorithms for conformal prediction include Support Vector Machines (SVMs) and k-Nearest Neighbours (k-NN).[3][1]

2.4.2 Inductive Conformal Prediction

One important drawback of Transductive Conformal Prediction (TCP) is the fact that it is not computationally efficient. When dealing with a large amount of data, it is inadequate to use all previous examples to predict an outcome for each new example. Hence, this approach is not suitable for any time consuming training tasks such as deep learning models. Inductive Conformal prediction (ICP) is a method that was outlined in [11] to solve the computational inefficiency problem by replacing the transductive inference with an inductive one. The paper shows that ICP preserves the validity of conformal prediction. However, it has a slight loss in efficiency.

ICP requires the same assumption as TCP (the i.i.d. assumption or the weaker assumption exchangeability), and can also be applied on any underlying machine learning algorithm. The difference between ICP and TCP consists of splitting the original training data set $(z_1, \ldots, z_n)$ into two parts in the inductive approach. The first part $D^{tr} = (z_1, \ldots, z_l)$ is called the proper training set, and the second smaller one $D^{cal} = (z_{l+1}, \ldots, z_n)$ is called the calibration set. In this case, the non-conformity measure Al based on the chosen underlying algorithm is trained only on the proper training set. For each example of the calibration set $i = l+1, \ldots, n$, a non-conformity score α_i is calculated by applying (4) to get the sequence $\alpha^{l+1}, \ldots, \alpha^n$. For a new example x_{n+1}, a non-conformity score α^y_{n+1} is computed for each possible $y \in Y$, so that the p-values are obtained and compared to the significance level ϵ to get the predictions such as:

$$\Gamma^{\epsilon}(x_{n+1}) = \{y \in Y : \frac{|\{i = l+1, \ldots, n, n+1 : \alpha_i \geq \alpha_{n+1}^{y}\}|}{n-l+1} > \epsilon\}.$$. ……(7)

In other words, this inductive conformal predictor will output the set of all possible labels for each new example of the classification problem without the need of recomputing the non-conformity scores in each time by including the previous examples, i.e., only α_{n+1} is recomputed for each y in Eq. (7).[3][1]

2.4.3 Density-Based Conformal Prediction

The paper [6] uses a density-based conformal prediction approach inspired from the inductive approach and considers a density estimate $\hat{p}(x|y)$ of $p(x|y)$ for the label $y \in Y$. Therefore, this method divides labeled data into two parts: the first one is the proper training data $D^{tr}=\{X^{tr},Y^{tr}\}$ used to build $\hat{p}(x|y)$, the second is the calibration data $D^{cal}=\{X^{cal},Y^{cal}\}$ to evaluate $\{\hat{p}(x_i|y)\}$ and set $\hat{t}_y$ to be the empirical quantile of order ϵ of the values $\{\hat{p}(x_i|y)\}$:

$$\hat{t}_y = \sup\left\{t : \frac{1}{n_y} \sum_{\{z_i \in D_y^{cal}\}} I(\hat{p}(x_i|y) \geq t) \geq 1-\epsilon\right\}$$ ……(8)

where n_y is the number of elements belonging to the class y in D^{cal}, and $D^{cal}_y=\{z_i \in D^{cal} : y_i=y\}$ is the subset of calibration examples of class y. For a new observation x_{n+1}, we set the conformal predictor Γ^{ϵ}_d such that:

$$\Gamma_d^{\epsilon}(x_{n+1}) = \{y \in Y : \hat{p}(x_{n+1}|y) \geq \hat{t}_y\}.$$ ……..(9)

This ensures that the observations with low probability—that is, the poorly populated regions of the input space—are classified as $\emptyset$. This divisional procedure avoids the high cost of deep learning calculations in the case where the online approach is used. The paper [6] also shows that $|P(y \in \Gamma^{\epsilon}_d(x_{n+1})) - (1-\epsilon)| \to 0$ with $\min_y n_y \to \infty$, which ensures the validity of the model.[3][1]

3 Software Engineering Method

3.1 Muscle

Muscle is an MSA tool that can generate ensembles of high-accuracy alternative alignments. All replicates have equal average accuracy on benchmark test, including the MSA made with default parameters. By comparing results of downstream analysis (trees, structure prediction...) on different replicates, one can assess the effects of alignment errors on your study.[4][1]

Muscle Syntax:

muscle -in < input file > -out < output file >

From the above syntax

It takes the input files of any file recognized file formats such as ".txt" , ".pdf" , ".doc"

It generates the output files in various formats as per the user requirement. Some of the formats are ".aln" , ".fa" , ".afa"

3.2 HMMer

HMMER is used for searching sequence databases for sequence homologs, and for making sequence alignments. It implements methods using probabilistic models called profile hidden Markov models (profile HMMs).

HMMER is often used together with a profile database, such as Pfam or many of the databases that participate in Interpro. But HMMER can also work with query *sequences*, not just profiles, just like BLAST. For example, you can search a protein query sequence against a database with **phmmer**, or do an iterative search with **jackhmmer**.

HMMER is designed to detect remote homologs as sensitively as possible, relying on the strength of its underlying probability models. In the past, this strength came at significant computational expense, but as of the new HMMER3 project, HMMER is now essentially as fast as BLAST.

HMMER can be downloaded and installed as a command line tool on your own hardware, and now it is also more widely accessible to the scientific community via new search servers at the European Bioinformatics Institute.[4][2]

HMMer Syntaxes:

a) **hmmbuild - construct profiles from multiple sequence alignments**

Synopsis

hmmbuild [options] hmmfile msafile

Description

For each multiple sequence alignment in msafile build a profile HMM and save it to a

new file hmmfile.

msafile may be '-' (dash), which means reading this input from stdin rather than a

file.

hmmfile may not be '-' (stdout), because sending the HMM file to stdout would

conflict with the other text output of the program.

Options

-h Help; print a brief reminder of command line usage and all

available options.

-n <s> Name the new profile <s>. The default is to use the name

of the alignment (if one is present in the msafile, or, failing

that, the name of the hmmfile. If msafile contains more than

one alignment, -n doesn't work, and every alignment must

have a name annotated in the msafile (as in Stockholm #=GF

ID annotation).

-o <f> Direct the summary output to file <f>, rather than to stdout.

-O <f> After each model is constructed, resave annotated, possibly modified source alignments to a file <f> in Stockholm

format. The alignments are annotated with a reference annotation line indicating which columns were assigned as

consensus, and sequences are annotated with what relative

sequence weights were assigned. Some residues of the alignment may have been shifted to accommodate restrictions of

the Plan7 profile architecture, which disallows transitions

between insert and delete states.[4][2]

b) hmmpress - prepare a profile database for hmmscan

Synopsis

hmmpress [options] hmmfile

Description

Constructs binary compressed datafiles for hmmscan, starting from a profile database

hmmfile in standard HMMER3 format. The hmmpress step is required for hmmscan to work.

Four files are created: hmmfile.h3m, hmmfile.h3i, hmmfile.h3f, and hmmfile.h3p. The

hmmfile.h3m file contains the profile HMMs and their annotation in a binary format.

The hmmfile.h3i file is an SSI index for the hmmfile.h3m file. The hmmfile.h3f file contains precomputed data structures for the fast heuristic filter (the MSV filter). The

hmmfile.h3p file contains precomputed data structures for the rest of each profile.

hmmfile may not be '-' (dash); running hmmpress on a standard input stream rather

than a file is not allowed.

Options

-h Help; print a brief reminder of command line usage and all

available options.

-f Force; overwrites any previous hmmpress'ed datafiles. The default is to bitch about any existing files and ask you to delete them first.[4][2]

c) hmmscan - search sequence(s) against a profile database

Synopsis

hmmscan [options] hmmdb seqfile

Description

hmmscan is used to search protein sequences against collections of protein profiles. For each sequence in seqfile, use that query sequence to search the target database of profiles in hmmdb, and output ranked lists of the profiles with the most significant matches to the sequence.

The seqfile may contain more than one query sequence. Each will be searched in turn against hmmdb.

The hmmdb needs to be press'ed using hmmpress before it can be searched with hmmscan. This creates four binary files, suffixed .h3{fimp}.

The query seqfile may be '-' (a dash character), in which case the query sequences are read from a stdin pipe instead of from a file. The hmmdb cannot be read from a stdin stream, because it needs to have those four auxiliary binary files generated by hmmpress.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The --tblout and --domtblout options save output in simple tabular formats that are concise and easier to parse.

The -o option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

Options for Controlling Output

-o <f> Direct the main human-readable output to a file <f> instead of the default stdout.

--tblout <f> Save a simple tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target model found.

--domtblout <f> Save a simple tabular (space-delimited) file summarizing the per-domain output, with one data line per homologous domain detected in a query sequence for each homologous model.

--pfamtblout <f> Save an especially succinct tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target model found.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can

greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but

can truncate target profile description lines.

--textw <n> Set the main output's line length limit to <n> characters per

line. The default is 120.[4][2]

4 How to Use My Project

System Requirements: 2GB RAM or more, 2.2MHz Processor or more, Linux Operating System.

Step 1: Intalling HMMer

% wget http://eddylab.org/software/hmmer/hmmer.tar.gz

% tar zxf hmmer.tar.gz

% cd hmmer-3.3.2

% ./configure --prefix /your/install/path

% make

% make check

% make install

The HMMer tool gets installed by following the above commands

Step 2: Install muscle

./muscle3.8.31_i86linux64

Specify the location of the file that is downloaded in linux

Open the muscle file using "./"

Step 3: Run the shell script as provided in the project file

rm -rf *out*

Removes files named out in their naming convention. It is used in case of any duplicate files that may cause issues while executing the project

./muscle3.8.31_i86linux64 -in PF00069_seed.txt -out PF00069_muscle_out

Creates a Multiple Sequence Alignment file for the given dataset. PF00069_seed.txt contains proteins of pikanese family.

hmmbuild PF00069_hmmbuild_out.hmm PF00069_muscle_out

For the generated muscle file, a HMM file is created by using hmmbuild command

hmmpress PF00069_hmmbuild_out.hmm

After creation of hmm file, in order to derive the metrics of the generated hmm file, it is pressed so that it ready to perform complex hmm operations on it like scan, search, pgmd etc.

hmmscan --tblout PF00069_hmmscan_out PF00069_hmmbuild_out.hmm PF00069_seed.txt

The sequences from the input file are sent to the hmm file in order to obtain the results. The results/ metrics generated are stored in the scan_out file.

hmmsearch --tblout PF00069_hmmsearch_out PF00069_hmmbuild_out.hmm PF00069_seed.txt

The sequences from the input file are sent to the hmm file in order to obtain the results. The results/ metrics generated are stored in the search_out file.

grep PF00069_muscle_out PF00069_hmmscan_out | awk '{ print $5 }' >> E0_value_out.txt

The scan out file contains various metrics of the sequences and this project mainly concentrates on the “E value” parameter. Hence we obtain only the E value by using grep command in linux

mkdir singleoutfiles

To create a directory of all the outfiles generated in the next steps

rm -rf ./singleoutfiles/*

To remove any files that many cause issues to the script execution.

wc=$(wc -l < ./PF00069_seed.txt)

counts the number of lines in the file PF00069_seed.txt

for i in $(seq 1 7 $wc);

do

    let inc=$i+6;

```
let quit=$inc+1;

sed -e "$i","$inc d" PF00069_seed.txt >> ./singleoutfiles/PF69_exclude_out$i.txt
```

The sed command is used in order to remove each sequence from the file alternatively and generate a new file

```
./muscle3.8.31_i86linux64 -in ./singleoutfiles/PF69_exclude_out$i.txt -out ./singleoutfiles/PF69_muscle_out$i
```

For all the input files created are fed into Muscle in order to create MSA files

```
chmod ugo+rwx ./singleoutfiles/*
```

Linux command to give access permissions for newly generated files. (Sometimes it generates error for files access permissions. Hence it is a safe method to avoid errors)

```
Hmmbuild ./singleoutfiles/PF69_hmmbuild_out$i ./singleoutfiles/PF69_muscle_out$i
```

For every MSA file generated, a HMM file is created

```
chmod ugo+rwx ./singleoutfiles/*
```

Linux command to give access permissions for newly generated files. (Sometimes it generates error for files access permissions. Hence it is a safe method to avoid errors)

```
hmmpress ./singleoutfiles/PF69_hmmbuild_out$i
```

After the hmm are created, they are pressed so as they are ready to get operated for other commands

```
chmod ugo+rwx ./singleoutfiles/*
```

Linux command to give access permissions for newly generated files. (Sometimes it generates error for files access permissions. Hence it is a safe method to avoid errors)

```
sed -n "$i","$inc p" PF00069_seed.txt >> ./singleoutfiles/PF69_seq_out$i.txt
```

Sed command seprates individual sequence and removes the rest of the sequences

```
chmod ugo+rwx ./singleoutfiles/*
```

```
# Linux command to give access permissions for newly generated files. (Sometimes it generates error for files access permissions. Hence it is a safe method to avoid errors)

        hmmscan --tblout ./singleoutfiles/PF69_hmmscan_out$i ./singleoutfiles/PF69_hmmbuild_out$i ./singleoutfiles/PF69_seq_out$i.txt

# hmmscan command feeds individual sequence generated above and feeds into the hmm file to obtain the metrics. Hence an E value is generated for an individual sequence.

        chmod ugo+rwx ./singleoutfiles/*

# Linux command to give access permissions for newly generated files. (Sometimes it generates error for files access permissions. Hence it is a safe method to avoid errors)

     grep PF69_muscle_out$i ./singleoutfiles/PF69_hmmscan_out$i | awk '{ print $5 }' >> ./singleoutfiles/Ei_value_out.txt

# For individual sequences fed into the hmm file, metrics are obtained in the scan out file. Grep command extracts only the Evalue of every sequence

done



wc=$(wc -l < ./E0_value_out.txt)

sum=0

for i in $(seq 1 1 $wc)

do

            z1=$(sed -n "$i","$i p" E0_value_out.txt)

# Prediction set of Label Y

                #echo $z1

                z2=$(sed -n "$i","$i p" ./singleoutfiles/Ei_value_out.txt)

# Prediction set for label Y^

                #echo $z2
```

```
                diff=$(awk -v a=$z1 -v b=$z2 'BEGIN{print (a-b)}')

                echo $diff >> diff.txt

                #echo " difference is $diff"

                sum=$(awk -v x=$sum -v y=$diff 'BEGIN{print (x+y)}')



# The step for conformal prediction

                #echo "sum is $sum"


done

echo $sum


Px=$(awk -v m=$sum -v n=38e0 'BEGIN{printf ("%1.5e\n"), (m/n)}')

# Step to compute the scientific notation of the p value

echo "P value is $Px"
```

4.1 Conclusion:

The data set with protein sequences is taken “PF00069_seed.txt”. It is fed into muscle and an MSA is generated. Later that MSA file is fed into HMMer and a hmm file is generated. The HMM file is pressed and scanned in order to obtain a metric called E value. E value describes how likely a sequence is related to the HMM file. Hence Y label is obtained with different conformity scores for conformal prediction.

The same data set is operated differently in the next step. A sequence is eliminated from the 38 sequences and an MSA file is generated for the remaining 37 sequences. HMM file is created and the removed sequence is fed into the hmm file and E value is obtained. This process is repeated for all the 38 sequences in the file and 38 E values are obtained. Hence the Label $Y^{\wedge}$ is obtained with different conformity scores for conformal prediction

As the algorithm of Conformal Prediction, we compare the values of Labels Y and Y^ and probability is determined. As the score are very low and the values are described in the scientific notation (for example 5.3e-78) ,According to the density method of conformal prediction every value is compared with its subsequent value in the other set of labels by removing the difference. The difference of each consequent value is obtained and average value for the difference is computed for those differences.

This is the final step for conformal prediction with both the conformity scores. The obtained P value , best describes that the sequences of protein family are more likely suitable for the generated hmm files.

A novel approach is presented for model selection and test sample prediction using the non conformity score in the conformal prediction framework. The preposed model is both consistent and gives consistent generalization performance. This is the most cost efficient as well as time efficient method for researchers working in bioinformatics(predicting the protein sequences). Therefore, the nonconformity measure approach gives us a general way of choosing to make predictions, allowing us the flexibility to apply it to algorithms that are not based on large margins. In future work, we aim to investigate the applicability of our proposed model selection technique to other learning methods.

5 References:

[2][1]: Wikipedia

[2][3]: Encyclopedia of Bioinformatics and Computational Biology, 2019

[3][1]: Deep Conformal Prediction for Robust Models , (Soundouss MessoudiEmail authorSylvain RousseauSébastien Destercke)

[4][1]: Muscle Documentation

[4][2]: HMMer Documentation