Protein classification and domain detection with Probabilistic Suffix Trees

Grégoire Lejay

June 10, 2010

Contents

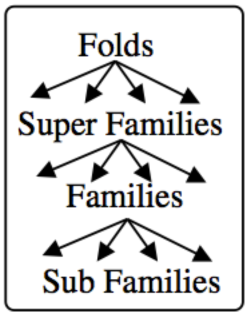
Abstract

Nowadays, the number of biological data is growing exponentialy and consequently the softwares made to cope with them need to be as fast that they are accurate. Here we will explain and evaluate the capacities of a method of protein classification using Probabilistic Suffix Trees models. We will also, talk about scoring methods that can increase the accuracy of those classifications considering domains among the protein.

1 Proteins and their classification

Products of the genetical code, proteins are organic compounds made of amino acids arranged in a linear chain and folded into a globular form. They are involved in almost every activities of the cell from atoms to the development of an organism.The term function of a protein rtefers to both aspects: to the biochemical function it has individualy or relatively to other molecules as assembly or complex. There are four main functions associated with proteins:
The first one is the binding, the ability to recognize other molecules. The myoglobin which fixes the oxygen molecule to iron atoms.An other one is the catalyze, the ability to trigger and accelerate chemical reaction.The DNA polymerase that catalyze the DNA replication is one example. They can be switches, this means that their shape can change under particular conditions and have a different function under different conditions. Finally, they can have a structural function inside the cell. Protein subunits can associate them-self with other proteins. This allows the construction of more complex systems and the definition of structures that shapes the cell. For example, the F actin associates with other related proteins to build the cytoskeleton. An important fact is that the shape and the chemicals properties of the amino acids which the proteins are composed of determine the function of a protein.(Petsko and Ringe)
Despite the fact that each protein possess its own tridimensional structure,many different proteins adopt the same shape, or architecture, a combination of secondary structure elements or domains(compact region of a the structure of the protein which is composed of continuous segment of amino acids and which is able to fold in sufficiently stable manner to be in an aqueous solution),this defines a first general group to classify proteins called **folds**. Thus, if two proteins fold in the same manner, we can suppose that they belong to a same set called a **superfamily**, if they may be homologous. This set can also be divided into **families** if some functions of the members of a superfamily are sufficiently close and if their similarity is greater

than 50 percent. Finally, if proteins within the same family perform similar functions but with small differences, they be classified into a group called **sub-families**.The following picture summarizes the hierarchy of the protein world.(Bejerano,Phd Thesis).



2 How to represent a family

A protein family is a group of evolutionary related protein sequences that exhibit many identical short statistical patterns of amino acids. Considering the evolution process, we can suppose that every protein family members descend from a single ancestral sequence. Thus, use statistical model can be a good idea to represent a group of related objects. Here, we will first present a method to represent a protein family and classify proteins : the profile Hidden Markov Models. Then, we will present the Probabilistic Suffix Tree the object we are interested in.

2.1 Profile Hidden Markov Models

2.1.1 How does it work ?

A profile hidden Markov Model describes a series of observations generated by hidden Markov Process where each state emits an observed symbol with a probability. In the case of proteins, the state of the Markov process are positions and the observations are amino acids drawn according to a multinomial distribution.
To represent a family, we need to build a profile-HMM that identifies a set of positions that describe the conserved subsequences of the family we consider. Let Fam be a family of protein sequence, the common structure of all the sequences within Fam can be view as a sequence of positions where amino acids occur. For each of these positions a probability distribution can be

defined over each amino acids that represents the probability of an amino acids to appear at this particular position and the probability that there is a gap at this position.
Thus, a profile HMM can be build with three different kind of states : the first one are the match states ($M = \{m_0, m_1, \cdots, m_n\}$), the second one are the insert states ($I = \{i_0, \cdots, \imath_{n-1}\}$) and the third one are the delete states ($D = \{d_1, \cdots, d_{n-1}\}$). There is a unique starting state m_0, a unique ending state m_n, the transitions always go from left to right such that a match or a delete are never visited more than once and for each state there are three possible transition.Delete states emit a symbol of deletion with a probability of one. Match and insertion states have different probabilities distributions, this is important to differentiate random to family related proteins.(Krogh et al.)
An example of profiles HMM:

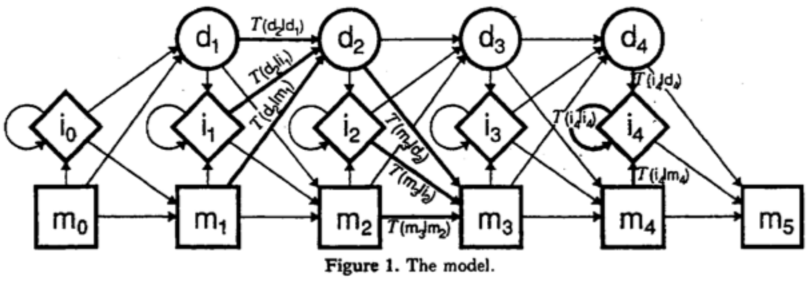


Figure 1. The model.

Three values can be computed (Ewens):
The first one is the evaluation problem ,compute the probability that $Model$ generates the sequence of observations O. It can be computed thanks to the forward algorithm in $O(N^2T)$ where N is the number of states and T is length of the sequence.
The second one is the decoding problem, given a sequence of observations O, find the most probable sequence of corresponding states. These can be done using the Viterbi Algorithm in $O(TN^2)$.
Finally, the learning problem, find the model that represent a given series of observations. In order to model a family of related sequence, the goal is to find the parameters (i.e.. transition and emission probabilities) that fit with the statistical properties of the considered family, indeed we can easily see that each set of parameters produce different set of related sequence, thus different families. This mean solve the learning problem and this can be done thanks to the Baum-Welsh algorithm or a multiple sequence aligment.
The main drawbacks of this method are if a particular transition occur in the family and if that transition is not in the training set it would be set to a zero probability transition in the HMM so they are very sensitive to the training

set use to build them. The number of parameters needed to build it are also a problem and slow down the computation time. Moreover, HMM profiles are sometimes trained thanks to multiple sequence alignments(see the exemple of Pfam database) that won't necessary give an optimal alignment.

2.1.2 Example of use: the Pfam database

Pfam is a protein families and domains database available on the internet(Bateman). Each family is represented by multiple sequence alignment and hidden Markov models. For each domains a set of sequences is selected and are put into a multiple alignment. This alignment will be use to train a HMM without the need of the Baum-Welsh algorithm.In fact, one can use multiple alignments to train an HMM, for every column of the alignment if it contains less than 50 percent of gaps then one can find the main state that will correspond to the column. If a column contains more than 50 percent of gaps, it will be an insert state. The corresponding emission distribution (amino acids distribution) can be computed thanks to the frequencies of each amino acids within the column. An alignment with a HMM is a path threw the model, this will be use to find the transition probabilities. So, if we compute the proportion of times these paths take a given transition in the multiple alignment, we can define the transition probabilities. These alignments are called *seed alignments*.
This is used to make the annotation of a query protein. As a profile HMM exists for each know domains and family,when a portion of the query has probability of having been generated by an profile HMM sufficiently high, the portion will be annotated by the domain represented by the profile HMM. The same kind of procedure is use for the families, if a query sequence matches with a family, a new seed alignment can be done and then a derived profile HMM, that's the way the Pfam-A database is built.
Unfortunatly, this is method is slow. In fact, the Pfam database is built thanks to the HMMer software that take about one or five minutes to train an profile HMM.In its early version, HMMer was said to take 100x much time than BLAST to find significantly similar sequence matches in a database.
Nowadays, HMMer third version is said to go as fast as BLAST. But it stills one of the major drawbacks of this method, thus PST are mean to do better.

2.2 The Probabilistic Suffix Trees

Many of the known approaches used to model a motif, a domain or a protein family starts by a multiple alignment which is not easy to do,and does not guarantee to find an optimal alignment. The idea is to identify short segments among protein sequences that reflects statistical properties of the corresponding family. Thus, we can compute a probability distribution on the next symbol to appear given the preceding subsequence by observing no more than a certain number of preceding symbols. So , a first idea could be to model those families with an order L Markov chain. But, one of the main drawback of the Markov chains is their trend to grow in an exponential way in the order of the chain. However, many natural phenomenon an event depends on past events contexts with variable length, speech is an example. So, it can be relevant to use variable length Markov chain. This one can be modeled by a structure called probabilistic suffix automaton which is equivalent to an other one called Probabilistic suffix tree.

2.2.1 The idea

So,what is a Probabilistic suffix Tree? A PST T over an alphabet Σ is a rooted tree where the degree of each node vary between zeros and $|\Sigma|$. Each vertex in the tree is associated with a symbol σ such that $\sigma \in \Sigma$ and that each path from a node v to the root define an unique sequence s. The node v is labelled by s.So, each parent node is the longest suffix of its children.
Moreover, each node in the tree is associated with a probability distribution which describes the probability $P^T(\sigma|s)$;the probability that the symbol σ appears given the longest subsequence of symbols that have been observed right before it (given s) . Let $\bar{\gamma}_s : \Sigma \to [0,1]$ be a function such that $\bar{\gamma}_s(\sigma) = P^T(\sigma|s)$, for each node in the PST T, we must have $\sum\limits_{\sigma\in\Sigma} \bar{\gamma}_s(\sigma) = 1$.
This distribution will be used by the algorithm to predict significant patterns among a sequence.

2.2.2 Prediction using a suffix tree

In order to know if an amino acid sequence s belongs to a family, we need to compute the probability P^T that s is generated by the PST corresponding to the family. And then, compare P^T with the probability that s is generated by an uniform distribution.
To compute the P^T, we need to travel s letter by letter. At each step, we look for the longest suffix θ that appears in the tree (θ is the label of a node of the tree) and that finish just before the last letter travelled.This represents the longest significant context influencing the probability of the next symbol.In fact, out of this context the probability of the next symbol is quite independent of all other symbol.

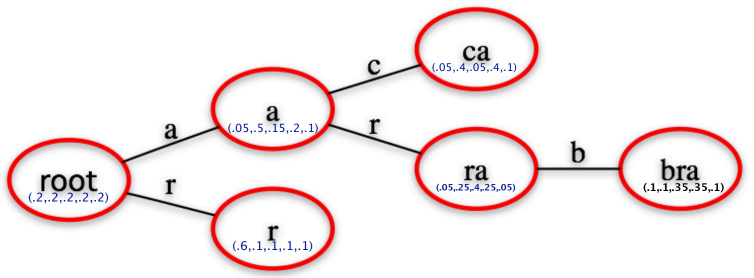
Thus, $P^T(s) = P^T(s_1).\prod_{i=2}^{|s|} P^T(s_i|(s_1 \dots s_{i-1}))$

In the right part of each member of $P^T(s)$, we find θ which corresponds to a node of the PST.

So, $P^T(s) = \bar{\gamma}_{root}(s_1).\prod_{i=2}^{|s|} \bar{\gamma}_{\theta}(s_i)$

where $\bar{\gamma}_s$ is the distribution associated with the node labelled by s.

For exemple: The prediction of the string s =abradacaba using the PST below:



$P^T(abradacaba) =$
$P^T(a).P^T(b|\underline{a}).P^T(r|ab).P^T(a|ab\underline{r}).P^T(d|a\underline{bra}).P^T(a|abrad).$
$P^T(c|abrad\underline{a}).P^T(a|abradac).P^T(b|abrada\underline{ca}).P^T(a|abradacab)$
$= \bar{\gamma}_{root}(a).\bar{\gamma}_a(b).\bar{\gamma}_{root}(r).\bar{\gamma}_r(a).\bar{\gamma}_{bra}(d).\bar{\gamma}_{root}(a).\bar{\gamma}_a(c).\bar{\gamma}_{root}(a).\bar{\gamma}_{ca}(b).\bar{\gamma}_{root}(a)$
$= 0, 2.0, 5.0, 2.0, 6.0, 1.0, 2.0, 15.0, 2.0, 4.0, 2$
$= 5, 76.10^{-6}$
The underlined sequences are the θ's found.
An uniform distribution would give a probability :

$P_{unfiform} = 0, 2^{10} = 1, 024.10^{-7}.$

So, $\frac{P^T(s)}{P_{uniform}} = \frac{5, 76.10^{-6}}{1, 024.10^{-7}} = 56, 25$

This make s be 56 times more plausible under the PST than under the uniform model.

2.2.3 How to build a PST

Here we present and explain the algorithm used to build a Probabilistic Suffix Tree. To do that we first need some definitions and notation which will be used to describe the algorithm.

Definitions and notations First, let $S_{sample} = \{r^i | i \in [\![1, m]\!]\}$ be a set (the training set) of strings over the alphabet Σ. In our context S_{sample} is the set of all the amino-acid sequences we want to use to produce a PST.
Each node of a PST is associated with a probability distribution which describes the probability that the symbol σ appears right after the sequence s,so few definitions are required to represent it:
We need an indicator variable $\chi_s^{i,j}$ which has value one if the string s such that $|s| = l$is a subsequence of the string r^i starting a the position j.
$\chi_s^{i,j} = \begin{cases} 1 \text{ if } s_1 s_2 \dots s_l = r^i_j r^i_{j+1} \dots r^i_{j+l} \\ 0 \text{ otherwise} \end{cases}$

Thus, we can can compute the number of occurrences of the string s in S_{Sample} with:

$$\chi_s = \sum_{i,j} \chi_s^{i,j}$$

. In order to have the empirical probability of observing the string s among S_{Sample} we need to compute the number of subsequences of the same length than s (l), it means compute :

$$N_{|s|} = \sum_{l_i \geq l} (l_i - (l - 1))$$

It is now easy to define the empirical probability of observing s by :

$$\tilde{P}(s) = \frac{\chi_s}{N_{|s|}}.$$

To compute the probability of a symbol σ to occur right after a given substring, we need to estimate a conditional probability.
$\tilde{P}(\sigma|s) = \dfrac{\chi_{s\sigma}}{\chi_{s\star}}$, where $\chi_{s\star}$ is the sum over every symbols α in the alphabet of the number of occurrences of the string $s\alpha$ in S_{Sample}.

The algorithm Here we present the algorithm which is the of major importance in this report. This will allow the construction of a PST given the a set of sequences. The PST model identifies significant appearance of short segments among many of the input protein sequence regardless of the position of these segments in the proteins sequences of the training set. These segments reflect some statistical properties of the protein family (the set) and are important to be well represented by the model; this can be controlled by the parameters leading the construction of the PST that represents the family. The algorithm presented below is a naive version for easy understanding of the building PST procedure, in fact it has been rafined and thus can be build in linear time and space (see Bejerano and Apolostico).

The first step is the initialization step:

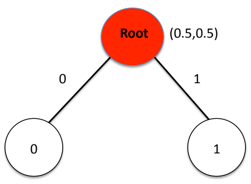
Let $\bar{T}$ be the PST. At the beginning, it will only consist of a single root node labelled by the empty word. And let $\bar{S} \leftarrow \{\sigma | \sigma \in \Sigma$ and $\tilde{P}(\sigma) \geq P_{min}\}$, this is the set of all the symbols that have a sufficient chance to appear in a sequence,later we will see that $\bar{S}$ is the candidates set, the set of the nodes that have a chance to be add to the tree.

The second step is the main loop of the algorithm:

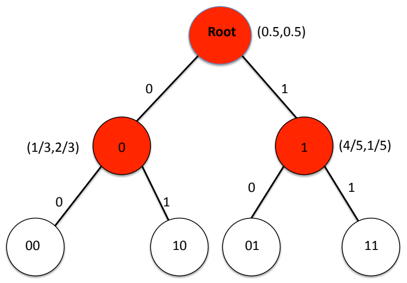
While the set $\bar{S}$ is not empty, pick $s \in \bar{S}$ and:
Remove s from $\bar{S}$.
If there exists a symbol $\sigma \in \Sigma$ such that:
$\tilde{P}(\sigma|s) \geq (1+\alpha)\gamma_{min}$
and
$\frac{\tilde{P}(\sigma|s)}{\tilde{P}(\sigma|suf(s))} \geq r$ or $\leq \frac{1}{r}$
Then add to $\bar{T}$ the node corresponding to s and all the nodes on the path to s from the deepest node in $\bar{T}$ that is a suffix of s.
This means that if s is in the $\bar{S}$ set, the probability that s occurs in the sequence is non negligible. So, if there exists symbols such that the probability of being observed right after s is also sufficient and such that the probability of being observed right after s is different of the probability of observing this symbol right after $suf(s) = s_2 \cdots s_l$ (this corresponds to the direct father of that node) then we can a new node to the tree labelled by s. Otherwise, the two probabilities are similar and that node contains no new informations, so it do not need to be add to the tree.
To end the loop, the algorithm must update the set of candidates.
Finally,the algorithm must assures that no symbol in the tree is predicted to have a zero probability.In the same time, it will compute the probability distribution associated to each node.
For each string s labeling a node in the tree, we compute:
$\bar{\gamma}_s(\sigma) = (1 - |\Sigma|\gamma_{min})\tilde{P}(\sigma|s) + \gamma_{min}$

Example If we follow the algorithm proposed by Gill Bejerano and Golan Yona with the following parameters $P_{min} = 0.1, \alpha = 0, \gamma_{min} = 0.1$ (the others have no real importance in this short example), we can easily build a PST from a short set of sequences.

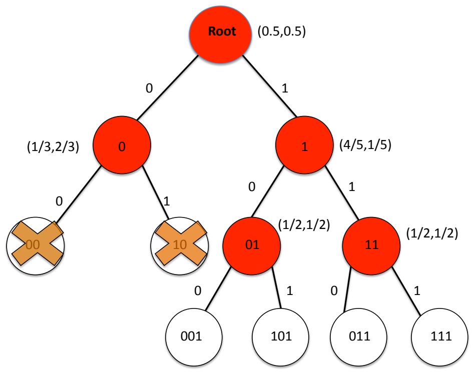
Let $A = \{0111, 1001, 0101\}$ be the set of sequences which will be represented by a PST. The algorithm starts with at the root (ϵ) and labels it with an uniform distribution. $a_i = \frac{1}{|\Sigma|}$.
Then it computes the empirical probability of observing 1 or 0 right after the empty word ($\tilde{P}(0|\epsilon) = \frac{5}{12}$) and ($\tilde{P}(1|\epsilon) = \frac{7}{12}$) those probabilities are not negligible, we can add the 0 and 1 nodes as candidates (white nodes).
At this step the tree looks like that:



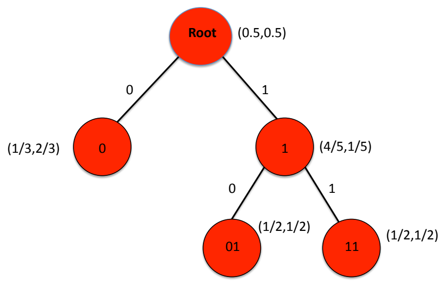
Now we compute for each node the corresponding distribution which is for each letter in the alphabet the probability of observing this letter right after the word that labels the node.
Thus, we compute:
$\tilde{P}(0|0) = \frac{1}{3}$
$\tilde{P}(1|0) = \frac{2}{3}$
$\tilde{P}(0|1) = \frac{4}{5}$
and $\tilde{P}(1|1) = \frac{1}{5}$.
Those two new distribution: $(\frac{1}{3}, \frac{2}{3})$ for the node 0 and $(\frac{4}{5}, \frac{1}{5})$ for the node 1 are different of their father node, so they are validated. An other step is to compute for each possible child of the considered nodes the empirical probabilities to observe them in the set of sequences.
We get: $\tilde{P}(00) = \frac{1}{9}, \tilde{P}(10) = \frac{2}{9}, \tilde{P}(01) = \frac{4}{9}, \tilde{P}(11) = \frac{1}{9}$, they are all greater than P_{min} so they are added to the tree.
We obtain the following tree:



Now, it's a new iteration of the algorithm and we compute the same things for all the nodes recently added to the tree.
Here, the nodes 00 and 10 are not validated because $\tilde{P}(0|00) = 0$ and $\tilde{P}(1|10) = 0$.However,the nodes 01 and 11 are validated because their distributions both equals to $(\frac{1}{2}, \frac{1}{2})$ that is different of $(\frac{4}{5}, \frac{1}{5})$ (the distribution of 1). Their child have all a sufficient probability of being observed in A, thus they added to the tree.



Finally, no of the last nodes added to the PST are validated because the strings : 0100, 1100, 0010,0001,1101,1011 and 1111 never appears in A.
At the end, the PST that represents A looks like that:



2.2.4 Complexity

If we let n be the length of the training set,the maximal depth of the PST L and the length of the query sequence m, a PST can be learned in $O(Ln^2)$ in time and $O(Ln)$ in space as there are $O(Ln)$ different subsequences of length between 1 and L. The prediction can be done in $O(Lm)$ because every symbol predicted requires traversing the tree from the root the a node of depth L.
This can be done in linear time with the sum of all sequences in the training set for the learning step and with the number of residues for the the prediction step. (Bejerano,Apolostico).Moreover, the algorithm had another improvement using the Adaptive VOMC Alogrithm (see Marcel Schulz et al.) and this is the version we use. Thoses differences lead to differences for the running time which faster than the non rafined version of the algorithm and for the choice of the parameters use for the training step.

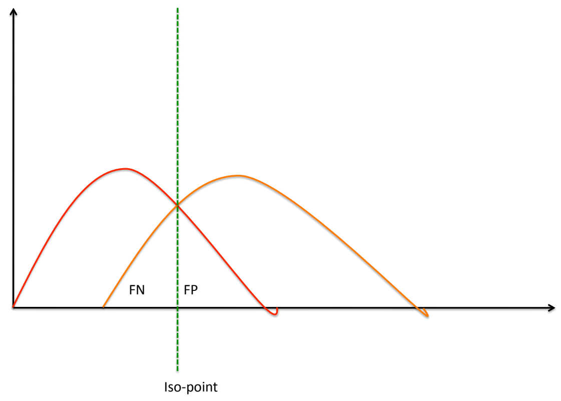
3 Family classification : Methods and results

3.1 Method

To establish a classification, each family set of the Pfam database was cut in five and $\frac{4}{5}$ of the family set was use to train a PST and then use to classify the protein sequences in the SwissProt database. The goal is to decide if a sequence in a protein database belongs to the family we consider or not. Such discriminative process is called a binary classification. So, for each sequence in the database, we will compute the probability that this sequence is generated by the PST that represent the family divided by the length of the sequence. Because short sequences should have a high probability and long sequence a low probability, this normalization is important to avoid bias due to length difference.

3.1.1 Equivalence number criterion

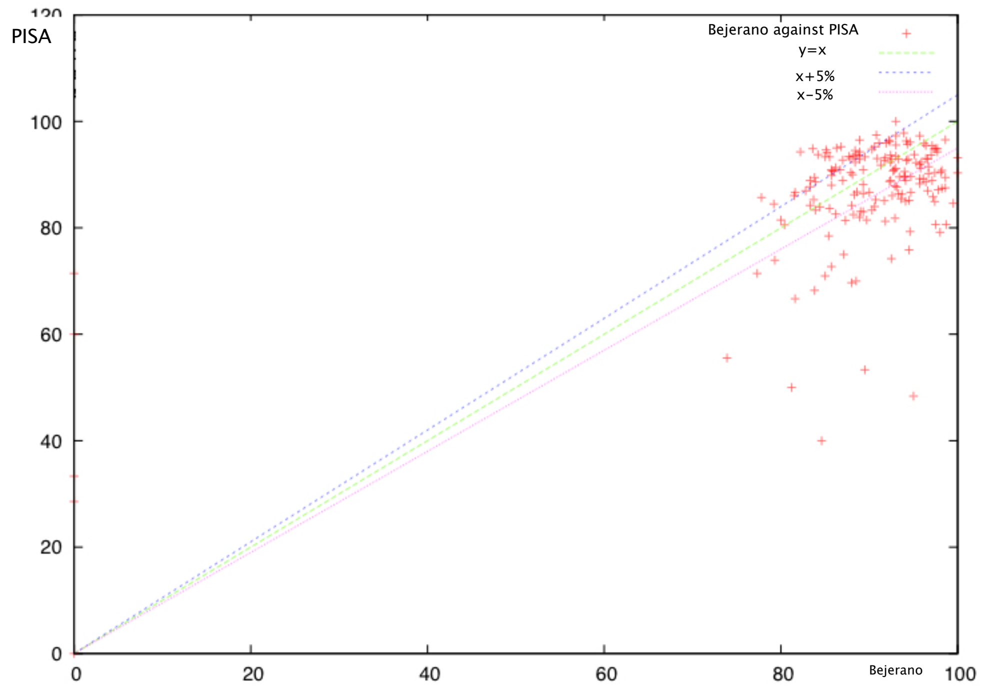
This is a method to estimate the accuracy of a PST. First, one will compute a score for every sequence in the database and a score for every sequence in the training set. The equivalence number criterion procedure will find the point (iso-point) where the number of False Positives equals the number of False Negatives. This also mean to find the point where the number of protein that doesn't belong to the family whose probability is a above a threshold and the number of proteins that belong to the family but with a probability less than the threshold. In that case, a decision rule is quite easy to determine. In fact, if a sequence belongs to the family and its probability is above the threshold then we can consider that the PST detected the sequence. The iso-point is the point where the number of False Positives (FP) equals the number of False Negatives (FN); then it is also the point where the specificity (the ability to reject sequences that doesn't belong to the family : $\frac{TN}{TN+FP}$) and the sensitivity (the ability to detect sequences that belong to the family :$\frac{TP}{TP+FN}$) are well balanced.



3.1.2 Two different kinds of scores

Here we present two different way to score a sequence given a PST an we will compare the results to those obtained by Bejerano and Yona. In fact, the implementation of the PST we use is not the original one programmed by Bejerano and Yona, and contains some important differences we will approach in the section concerning the determination of the parameters.

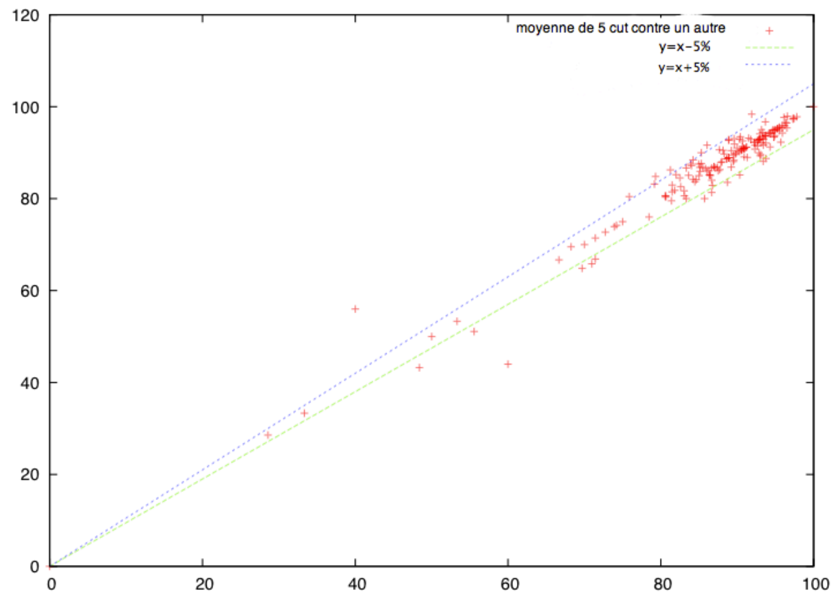
The likelihood: This first method is the one explained in the Prediction using a probabilistic suffix tree. The goal is to compute for each sequence in the SwissProt database the probability that this sequence is generated by the PST that represents the family. And then compute the sensitivity of our PST thanks to the equivalence number criterion.
After computing and finding a good set of parameters for each 175 families of the first version of Pfam,we computed the sensitivity of the trained PST and we made a scatter plot to compare the Bejerano's results and the ones got with PISA toolbox(on x axes Bejerano's results, on y axes PISA).



Scatter plot of Bejerano's results against PISA,and a marging of 10percent.

Now, one can observe that the results are well balanced, there are 38 points in the margin, 65 point Bejerano's results advantage and 72 in Pisa's advantage. A zero sensitivity on bejerano's results are due to missing data. Low results for PISA's usually correspond to family with a short number of sequences whose have a lot of residues (like lamininEGF),thus it will be hard to tune correctly the parameters. In our computation, we didn't let

the values of the maximum depth be larger than 20, it can be good for some families to use a greater value of depth.
To test if the results wasn't only due to choice of the training set, the same procedure was made with 6 different cut. The following scatter plot shows the mean of five of those results (on x axe) and one particular set of results that wasn't use to compute the mean(on y axe).We can see,the quality of PSTs is not very sensitive to the training set use to build it because most of the results are in a margin of 10 percent, this is also one of the qualities of PST that make them interesting.

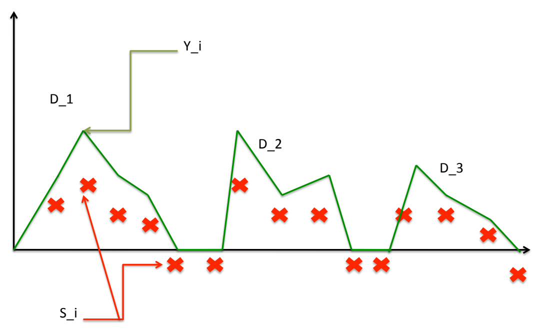


The local score: In their article Gill Bejerano and Golan Yona used a global method to their prediction, but this way contains some limitations. In fact, the decision can be deceived if a sequence is quite similar to another family member. The global prediction will miss local similarities that could be specific to the family, thus the probability can lowered by long unrelated part of the sequence.
To do so, one will travel the query sequence amino acid per amino acid and compute the following value:

$$S_i = log(\frac{P(s_i|FamilyModel, s_{i-1}, s_{i-2}, \cdots, s_1)}{P(s_i|BackgroundModel, s_{i-1}, s_{i-2}, \cdots, s_1)})$$

Where, i is the position among the protein sequence, s_i is the amino acid at the position i, FamilyModel is the PST that represents the family, BackgroundModel is the PST that represents all the SwissProt database. This ratio will compare the probability of the two events : the amino acid s_i is

generated by FamilyModel or it is generated by the BackgroundModel.This mean compare the probability that s_i occurs in the family or occurs under independent selection.
Then the local score is compute that way :
$Y_i = max\{(Y_{i-1} + S_i), 0\}$
The idea is that this score will decompose the sequence in islets, each islet will correspond to a region of the sequence where the amino acids are more likely to be generated by FamilyModel than by BackgroundModel. We can suppose to be domains those islets,the value $a_i = maxY_i$ will be use to annotate it. The following drawing represents the idea of islet computed by the local score.



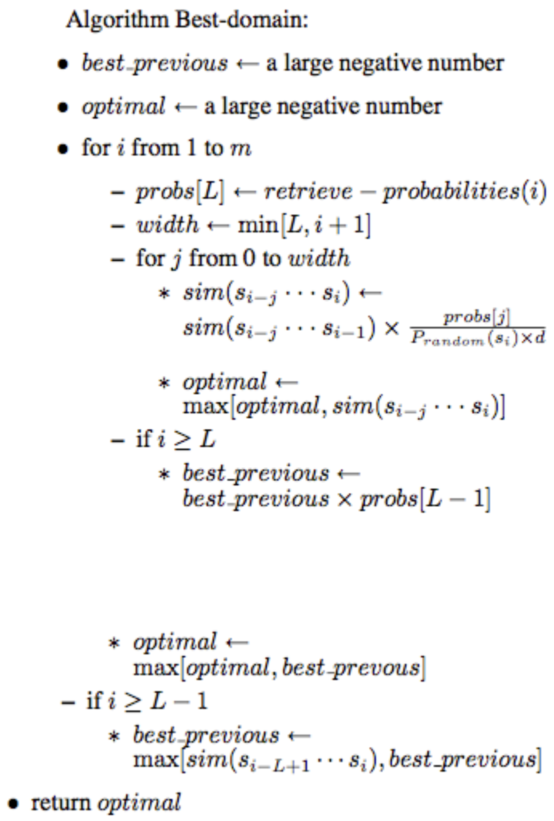
In the PISA's implementation the local score is not computed that way, and has important changes.

First, the values $S_i = log(\frac{P(s_i|FamilyModel, s_{i-1}, s_{i-2}, \cdots, s_1)}{P(s_i|BackgroundModel, s_{i-1}, s_{i-2}, \cdots, s_1)})$ are stored in a tree called score tree, which is a kind of union of the Background tree and the family tree and such that the distribution associated to each node is the values S_i.
!!PLus de precisions!!
Secondly, the local score computation is made differently. A first algorithm will find the most conserved region in the query sequence(Best-domain algorithm), and this will be use to make the classification. In section 4, we will present a variant of this algorithm that find one or more conserved regions (multiple-domain prediction).

Best-domain Algorithm The main idea of this algorithm is that if a region of sequence is conserved in a family, the probability of seeing a symbol at a position only depends on the symbols inside this region. This is the main difference with the global prediction approach which may compute probabilities given symbols outside the conserved region.
A first object need to compute this prediction is the similarity score, which is similar to the S_i's we saw before.

$$sim(s_i \cdots s_{j-1}s_j) = \frac{P(s_i \cdots s_j|FamilyModel)}{P(s_i \cdots s_j|BackgroundModel)}$$
$$= sim(s_i \cdots s_{j-1})\frac{P(s_j|FamilyModel, s_i \cdots s_{j-1})}{P(s_j|BackgroundModel)}$$

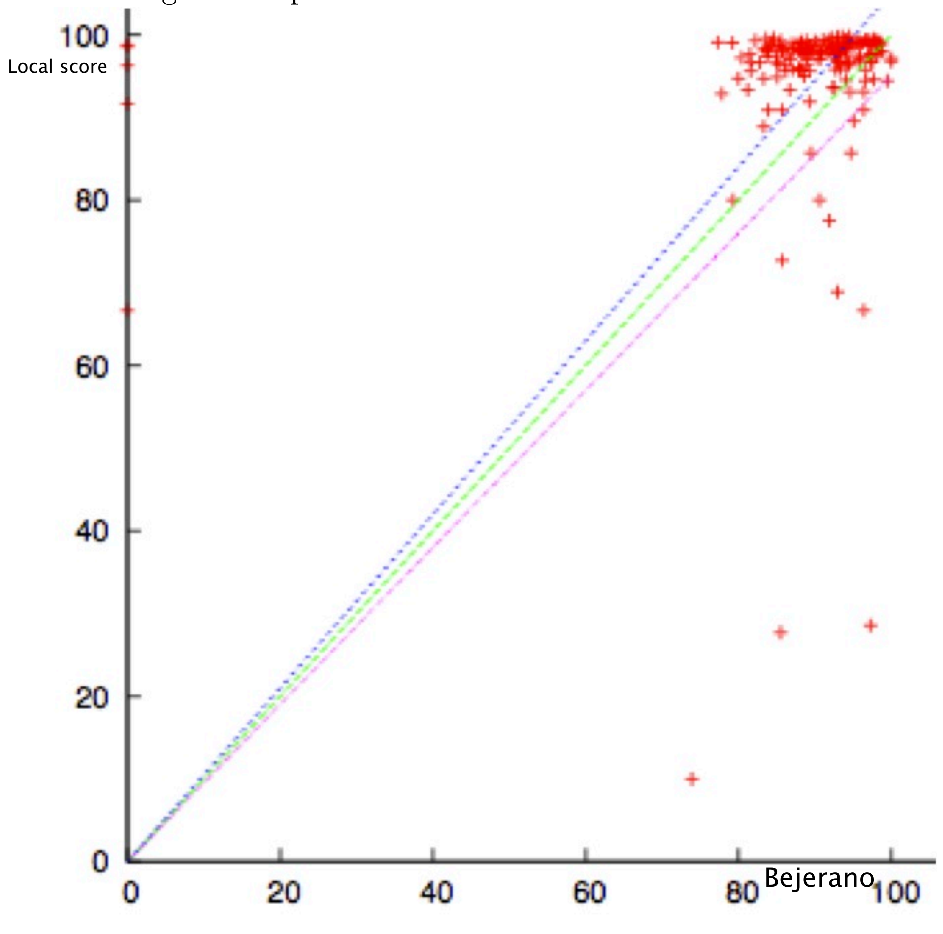
An other one is function able to traverse the PST and to retreive the probability of the symbol s_i for all subsequences. This mean compute the values : $probs[j] = P(s_i|s_j \cdots s_{i-1})$ for all $j \in [i - L + 1, i]$



```
Algorithm Best-domain:
• $best\_previous \leftarrow$ a large negative number
• $optimal \leftarrow$ a large negative number
• for $i$ from 1 to $m$
    – $probs[L] \leftarrow retrieve - probabilities(i)$
    – $width \leftarrow \min[L, i + 1]$
    – for $j$ from 0 to $width$
        * $sim(s_{i-j} \cdots s_i) \leftarrow$
          $sim(s_{i-j} \cdots s_{i-1}) \times \frac{probs[j]}{P_{random}(s_i) \times d}$
        * $optimal \leftarrow$
          $\max[optimal, sim(s_{i-j} \cdots s_i)]$
    – if $i \geq L$
        * $best\_previous \leftarrow$
          $best\_previous \times probs[L - 1]$
      * $optimal \leftarrow$
        $\max[optimal, best\_prevous]$
  – if $i \geq L - 1$
      * $best\_previous \leftarrow$
        $\max[sim(s_{i-L+1} \cdots s_i), best\_previous]$
• return $optimal$
```

The idea here is not to compute the similarities for the whole sequence but only for each each subsequences. This can be justified by the fact that if we want to compute the probability of a symbol inside a domain, it is more relevant to compute it starting to the beginning of the domain than from the beginning of the sequence.

As the likelihood classification, we trained a PST for each familly, compute a score for each sequence in the SwissProt database for each PST and use the equivalence number criterion to decide if a sequence belongs to a family or not.
Thanks to the local score, the overall sensitivity is improved of 7 percent. And the results are globally in Local score advantage relatively of the classical method,on 175 families 148 got a better sensitivity.
The following scatter plot summarize this.



However, one can notice some really bad results compared to the others. This is particularly the case for the wap and the zf-CCHC families. The wap family had in both likelihood and local score bad results (local score :27.8, likelihood 40) and is always among the worst cases, this is may be due to the fact this family is composed of only 13 sequences with a few number of residues. The case of the zf-CCHC family is different, this family contains a lot of sequences, and its bad results with the local score with is the best domain algorithm is due to the fact that this family possess a great variety of architectures and a lot of domains, thus the best domain algorithm may not be a good idea for this protein and the multiple domain algorithm will have better results.

3.1.3 Multiple domains prediction

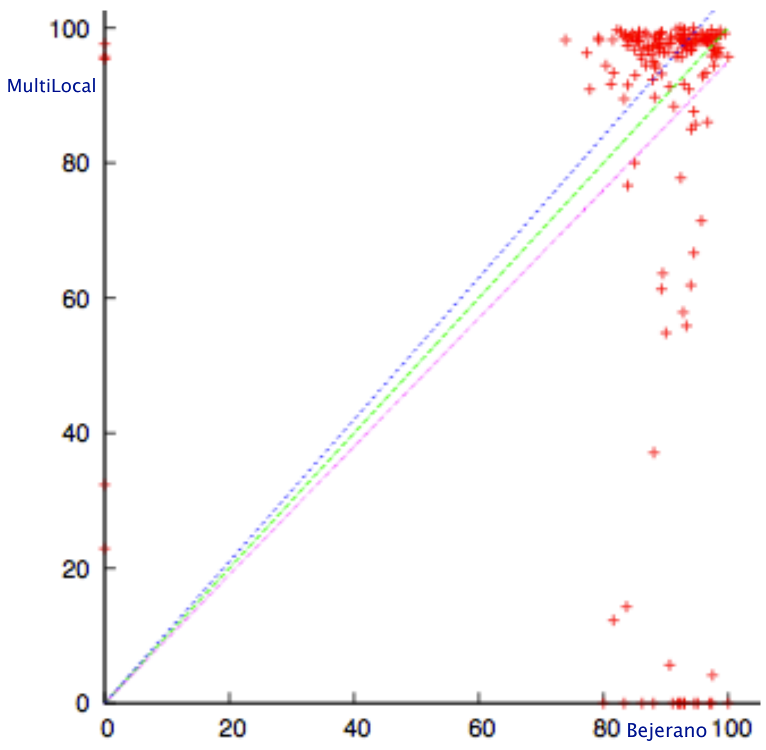
The idea As we seen before some bad results in the classification process may occur when families have a great variety of domains, thus we want an algorithm that consider the fact multiple domains may occur in a sequence. So, contrary to the previous algorithm that sum up the similarity score for the whole sequence, multiple domain algorithm will compute this sum for separated regions. The best sum will be use for the classification step using the equivalence number criterion. This will be done using dynamic programming, it will combine solutions of sub-problems in order to find an optimal solution of the problem.
To do this, four set of values will be used, the algorithm will travel the whole sequence,and compute:
•$maxBefore[i]$ will denote the best score for all the subsequences that ends at the position $i-L-1$ where L is the length of the longest context found for all the subsequences from 1 to i. $startBefore[i]$ is an index that represents the position of the subsequence that gave a maximum.
•$maxHere[i]$ which denote the best score at position i
•$maxSoFar[i]$ that denote the best score we get from 1 to i.

The algorithm Here is the multiple-domains algorithm presented by Zhaohui Sun , but in our implementation this algorithm is a little different because it must be adapted to the PISA implementation and the use of score trees.

```
1: $maxBefore[0] \leftarrow -\infty$
2: $maxHere[0] \leftarrow -\infty$
3: $maxSoFar[0] \leftarrow -\infty$
4: for $i = 1$ to $m$ do
5:     $probs[L] \leftarrow retreive - probabilities(i)$
6:     $maxHere[i] \leftarrow -\infty$
7:     $width \leftarrow minL, i+1$
8:     for $j = 0$ to $width$ do
```

```
 9:      $sim(s_{i-j} \cdots s_i) \leftarrow sim(s_{i-j} \cdots s_{i-1} * \frac{probs[j]}{P_{random}(s_i)} d)$
10:      $maxHere[i] \leftarrow max(sim(s_{i-j} \cdots s_i) * maxSoFar[i-j-1], maxHere[i])$
11:    end for
12:    if $(i >= L)$ then
13:      $maxBefore[i] \leftarrow maxBefore[i-1] * probs[L-1]$
14:      $maxHere[i] \leftarrow max(maxHere[i], maxBefore * maxSoFar[startBefore[i] - 1])$
15:      $maxSoFar[i] \leftarrow max(maxSoFar[i-1], maxHere[i] - T)$
16:    end if
17:    if $(i >= L-1)$ then
18:      $maxBefore[i] \leftarrow max(sim(s_{i-L+1} \cdots s_i), maxBefore[i])$
19:    end if
20:  end for
21:  return $maxSoFar[m]$
```

Results For this method,we launched the classification procedure without searching the best parameters because this algorithm is not linear and because the maximum values we obtain are too extremes, therefore some computers cannnot compute correctly with it, but this can be change in our implementation. The parameters used are those we obtained for the best-domain algorithm.



We can see that despite the fact that a lot of values are very bad,most of the the values are better than Bejerano's (119 in Multiple Domains advantage, 41 in likelihood advantage and 15 in the margin) and can be improved if the parameters are correctly tunned. For example the enolase family gave 0 as sensibility and can be improved to 99 of sensibility. The case of the zf-CCHC family we predict to have better results we this algorithm proved to be rigth, in fact the zf-CCHC gave 97.6 of sensibility without tunning the parameters. This is very encouraging.

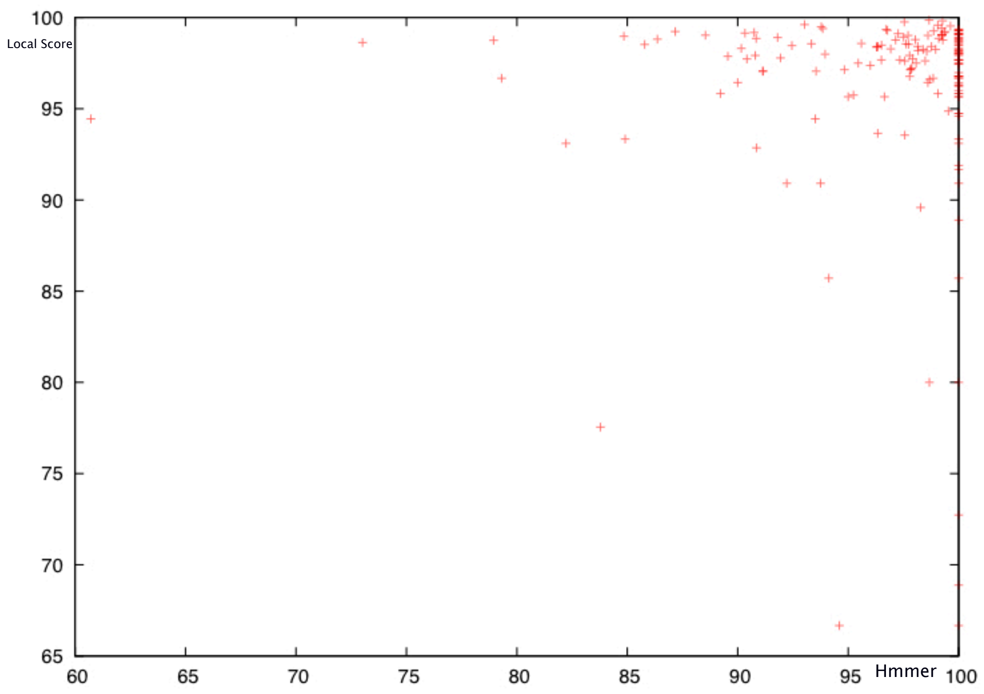
3.2 Determination of the parameters

Before to start the classification, one must find the best parameters to represent well the families. As there are five different parameters, it should be too long to test the sensitivity for every set of possible parameters. The procedure use to find a good set of parameters works as follow: first start with the set that used in his article that lead to bad results with the PISA implementation, in fact the parameter P_{min} works differently, the threshold it sets in the algorithm cannot be respected for each node because after the building of the suffix tree, it is then converted into a suffix trie. So, it must vary one of the parameters, and then find the parameters that give the best sensitivity. Then the same loop is applied but it keeps the parameter that

gave a maximum sensitivity. This method is naive but gave good results if the order of the parameters is chosen correctly, but sometimes some bad results occurs when the final combination of the parameters decrease the sensitivity.
The order of the parameters is important,several test were made with different order of evaluation and gave different results, the best found was :
$r(cutoff) \rightarrow L(depth) \rightarrow \alpha \rightarrow \gamma_{min} \rightarrow P_{min}$
In the case of the local score, the background model should have a low value for L (between 3 to 6) for the reason that if the tree is too big, there will be too many node in the score tree that will produce a negative score. The mean of the score of a random sequence is expected to be negative in the case of local score.

3.3 Comparaision with HMMer

Here we present a comparision between the local score with Best-domain algorithm and the results of the first version of HMMer on the first release of the Pfam database.
Despite the fact that three of the families that gave bad results with the local score were deleted from this plot (wap,lamininEGF, and zf-CCHC), we see that local score method is better but never give 100 percent of sensibility results.



4 Conclusion

In this report,we saw that the PISA's implementation of PSTs is better than the one made by Bejerano and Apolostico. We explain the idea of the local score using best domain algorithm and saw that its capacities are very promising relatively to the famous HMMer. Bad results can be easily improved thanks to the multiple domains algorithm which is a more general view of the Best dommain algorithm but a little slower.
However, it's a pity that we couldn't compare our results with the last version of Pfam because of its organization, this can be done if the organization of our source code is reviewed, in fact the Pfam database evolved a lot these last years and is fully adapted to the HMMer program.It should be also interesting to compare the local score with the last version of Hmmer which is said to be faster and more accurate than its previous versions.The fact that we couldn't find best parameters with the best domain algorithm was only due to time problems and computer capacities; it seems that it can be computed on a cluster. A backtacking process of the multiple domains algorithm could be programmed easily in order to acheive the annotation process.

5 Bibliography

•Gill Bejerano and Golan Yona,Variations on probabilistic suffix trees:statistical modeling and prediction of proteins families.
•Zhaohui Sun,Jitender S.Deogun,Local Prediction Approach for Protein Classification Using Probabilistic Suffix Trees.
•Warren Ewens,Gregory Grant,Statisical Methods in Bioinformatics: An Introduction.
•Petsko and Ringe,Structures and functions of proteins
•Gill Bejerano, Automata Learning and Stochastic Modelling for Biosequence Analysis, PhD Thesis.
•Anders Krogh,Michael Brown,I.Saira Mian,Kimmen Sjolander and David Haussler, Hidden Markov Models in Computational Biology, Applications to Protein Modeling.
• Bateman, A.; Coin, L.; Durbin, R.; Finn, D.; Hollich, V.; Griffiths-Jones, S.; Khanna, A.; Marshall, M. et al. (Jan 2004).The Pfam protein families database.
• Marcel Schulz, David Weese, Tobias Rausch, Andreas Doring, Knut Reinert, Martin Vingron. Fast and Adaptive Variable Order Markov Chain Construction.
• Apostolico, A., Bejerano, G.: Optimal amnesic probabilistic automata or how tolearn and classify proteins in linear time and space. J. Comput. Biol. 7(3-4).