Mining for Outliers in Sequential Databases

Pei Sun* Sanjay Chawla† Bavani Arunasalam‡

Abstract

The mining of outliers (or anomaly detection) in large databases continues to remain an active area of research with many potential applications. Over the last several years many novel methods have been proposed to efficiently and accurately mine for outliers. In this paper we propose a unique approach to mine for sequential outliers using Probabilistic Suffix Trees (PST). The key insight that underpins our work is that we can distinguish outliers from non-outliers by only examining the nodes close to the root of the PST. Thus, if the goal is to just mine outliers, then we can drastically reduce the size of the PST and reduce its construction and query time. In our experiments, we show that on a real data set consisting of protein sequences, by retaining less than 5% of the original PST we can retrieve all the outliers that were reported by the full-sized PST. We also carry out a detailed comparison between two measures of sequence similarity: the **normalized probability** and the **odds** and show that while the current research literature in PST favours the odds, for outlier detection it is normalized probability which gives far superior results. We provide an information theoretic argument based on entropy to explain the success of the normalized probability measure. Finally, we describe a more efficient implementation of the PST algorithm, which dramatically reduces its construction time compared to the implementation of Bejerano [3].

1 Introduction and Related Work

In many application domains an *interesting* event is defined in terms of its deviation from the norm. By definition such events are rare (otherwise they would become the norm!) and lend themselves to the proverbial "needle in a haystack" analogy. Examples of well known applications where the discovery of such events are important include network intrusion and fraud detection [10]. In fact one can argue that science in general progresses with the detection (often accidentally) of interesting events followed by an invention of new theories to explain them.

Within the data mining community a substantial body of research has developed to proactively search for interesting events or *outliers* in large databases. Starting from the folklore definition of outlier as an event "which is at least three standard deviations away from the mean", several novel and efficient methods have appeared in the recent past. In general, three main design patterns have emerged to detect and extract outliers based on *distribution, distance* and *density*. In the distribution-based approach the underlying statistical distribution of the data source is estimated, say M, and a data point d is considered to be an outlier if $P(d|M) < t$ for a user-specified threshold. A known limitation of this approach is that computing the distribution of complex heterogenous and high-dimensional data sets is non-trivial if not intractable. The distance-based paradigm was originally proposed by Knorr and Ng [9] in which each data point is represented as point in a n-dimensional space. Points whose distance to their k^{th} nearest neighbor is large are considered candidate outliers. Several variations and efficient algorithms on this have been proposed [2, 9, 11]. A limitation of distance-based outlier techniques is that they are not flexible to discover local outliers, especially in data sets which have non-uniform density as one moves across the data landscape. This limitation was lifted by Breunig et. al [6] who introduced the concept of Local Outlier Factor (LOF) which takes the local density into account when checking for outliers.

Most of contemporary approaches assume that the underlying data element can be represented as a relational tuple r where $r = (t_1, \ldots, t_n)$ and the $t_i s$ are either real-valued or categorical. For many application domains, data is most naturally represented as a sequence of symbols. For example, a piece of text can be perceived as a sequence of alphabets. Similarly in speech recognition, the sound wave is binned into a fixed set of categories, which then serves as an alphabet for the sound signal. The most prominent examples are the composition of the DNA and proteins. The DNA is a sequence from the alphabet set $\{A, G, C, T\}$. Similarly a protein is a sequence of amino acids from an alphabet of size twenty.

The challenge in sequence analysis is to define a notion of similarity, which can capture the structural differences between sequences. The edit distance, *edit*,

*The School of Information Technologies, University of Sydney.
The work of this author was supported by Capital Markets CRC.
†The School of Information Technologies, University of Sydney.
‡The School of Information Technologies, University of Sydney.

between two sequences s_1, s_2 is defined as the number of operations from the set {*insert, delete, replace*} that are required to map s_1 into s_2. Despite its widespread use, it is not hard to come up with an example set of sequences $\{s_1, s_2, s_3\}$, where s_1 should be more similar to s_2 than s_3 yet $edit(s_1, s_2)$ is greater than $edit(s_1, s_3)$. Moreover, edit distance can only be used as a similarity measure between two sequences, not between a sequence and a set of sequences. Another approach is to transform symbolic sequences into numerical or integral vectors, and then map sequences into points in a multi-dimensional space. Several different mappings have been proposed. Kahveci and Singh [8] have used wavelets, and Guralnik and Karypis [7] have used the frequent subsequences as "words" of the sequence and created an analog of the document-word matrix which is fed into a mining process.

The underlying statistical distribution of the sequences (in a given domain) is usually unknown and hard to estimate. However, there is a recurring property of sequences that manifests itself across domains, which the above methods do not exploit - at least explicitly. This property is called *short memory* by Ron et.al. [12] and is essentially a higher-order Markov condition. Namely, given a sequence $s = s_1 s_2 \dots s_l$, there exists an $L < l$ such that the conditional probabilities $P(s_l|s_{l-k} \dots s_{l-1}) \approx P(s_l|s_{l-L} \dots s_{l-1})\ \forall k > L$. Their key observation is that the length of the Markov chain in many domains is context-driven and not fixed. This led them to propose a variable-order chain, where the states of the system are variable length suffixes of sequences organized in a suffix tree. Associated with each node of the tree was a vector of transition probabilities from the state to the next symbol. They call the model a Probabilistic Suffix Automata (PSA), and they also introduce a variation on the PSA, which they call the Probabilistic Suffix Tree (PST). A PST can be represented more efficiently and allows for an efficient and approximate computation of the joint probability as

$$P^T(s_1 s_2 \dots s_l) = P^T(s_1)P^T(s_2|s_1) \dots P^T(s_l|s_1 s_2 ... s_{l-1})$$

We will describe the PST in more detail in Section 2. Bejerano et.al. [3, 5, 4] has used the PST to model biological sequences. They have provided an implementation of their algorithm, which we will compare against in the experiment section.

Yang and Wang [14] have introduced the CLUSEQ model to cluster sequences using the PST. The CLUSEQ model is quite powerful and allows for the number of clusters to be automatically set. The CLUSEQ model uses the odds measure to test for cluster membership, i.e., given a sequence $s = s_1 \dots s_l$ and a PST for cluster C, the membership of s in C is determined by the ratio of $P_C(s_1) \prod_{j=2}^{l} P_C(s_j|s_1 s_2 ... s_{j-1})$ and $\prod_{j=1}^{l} P(s_j)$. A ratio greater than one provides evidence that s belongs to C. Outliers can be detected using CLUSEQ because a sequence, which does not belong to any cluster, can be considered as a candidate outlier. However, our objective is to directly mine for outliers and we will make several observations, which makes it possible to do so more efficiently than going through the clustering route. If the objective is to directly find outliers, then the number of user-defined parameters that need to be set can also be reduced.

If the set of sequences is perceived as a configuration system, then intuitively the injection of outliers will increase the entropy of the system. Thus, one way of directly determining outliers in the system is to check if the removal of a sequence increases or decreases the entropy of remaining system. Indeed, this is one way of validating the discovered outliers. Furthermore, we will show that the similarity measure for outliers can be calculated by traversing nodes very close to the root of the PST. This will have implications in deciding the threshold values for pruning the tree. Finally, instead of using the odds similarity measure [4, 15], we will use the length normalized similarity measure [3]. Extensive experiments show that this measure is better suited for outlier detection.

The notations and basic concepts used in this paper are listed in Table 1.

1.1 Problem Definition 1 (P1)

Given: A set of sequences S and a number n.

Find: The top-n outliers in S.

1.2 Problem Definition 2 (P2)

Given: A set of sequences S, and a query sequence q and similarity threshold value t.

Determine: If q is an outlier with respect to S and t.

1.3 Key Insights and Contributions

We claim the following contributions towards the mining of outliers in a sequential database.

1. We describe a new and more efficient implementation of the PST algorithm compared to the one reported in [3]. Experiments on synthetic and real data sets show that our implementation is several orders of magnitude faster.

2. When a set of sequences are organized in a PST, the outlier sequences are found near the root of the PST, i.e., the maximal level we need to search

Name	Description
$\sum$	the alphabet
S	the string (sequence) set
$s = s_{1,l} = s_1 s_2 ... s_l$	a string (sequence) of length l, $s_j \in \sum\ j = (1, 2, ..., l)$
suffix of s	a segment s' of length l' if $s'_j = s_{j+l-l'}$ for $j = (1, ..., l')$
$P(s)$	empirical probability,$P(s) = \frac{the\ number\ of\ occurrences\ of\ s\ in\ S}{the\ maximal\ number\ of\ possible\ occurrences\ of\ string\ of\ length\ \|s\|\ in\ S}$
$P(s_j \| s_1 ... s_{j-1})$	the conditional probability of observing s_j right after $s_1 ... s_{j-1}$
PST	probabilistic suffix tree
$Pmin$	the minimal value of empirical probability, a threshold used to prune the nodes of a PST
$MinCount$	the minimal number of occurrence of a string in S, a threshold used to prune the PST
L	the maximal depth of a PST, a threshold used to control the depth of a PST

Table 1: Notations and basic concepts

in PST while computing their similarity values is small. By making use of this observation, we only need to construct part of the PST instead of the full one. Thus, we can reduce the cost (both running time and memory) if the intention is just to find outliers (Section 5.3).

3. The normalized probability similarity measure SIM_N (see Section 4) is a more suitable measure than the odds measure (SIM_O) for outlier detection. This is contrary to the current understanding of sequential similarity measures in the PST literature [3].

4. There is a strong correlation between the normalized probability measure and the entropy of the system. In fact, we note that by the Shannon-McMillan theorem $-SIM_N$ converges to the entropy of an information source (Section 5.7)

5. We will report on a wide array of experiments that we have carried out on synthetic and real data sets, which show that our overall approach is more efficient while retaining the accuracy of detecting outliers in a sequential setting.

1.4 Solution in a Nutshell

Our unified solution for P1 and P2 proceeds as follows.

1. Let S be a set of sequences from an alphabet of size $|\Sigma|$, where $|S| = N$ and the average length of sequences in S is m. Organize S into a PST of predetermined depth L. The construction of the PST requires several important parameters, which we will discuss in subsequent sections. For now, all we have to note is that a PST can be constructed in time $O(NmL) + O(L|\Sigma|^\alpha) + O(LC)$, where α is fixed integer, which depends upon the pruning parameters (α is usually less than 4) and C is a constant.

2. Once the PST is constructed the SIM_N of a sequence and the all sequences can be calculated in time $O(mL)$ and $O(NmL)$ respectively.

3. Sort the sequences using SIM_N in $O(NLogN)$ time.

4. Choose the sequences with the n lowest SIM_N values.

5. This is our proposed solution for P1. Thus, the cost of solving P1 is dominated by the construction cost (assuming $LogN < mL$).

6. Given a new query sequence q, the cost of calculating $SIM_N(q)$ is $(O(|q|L)$. We will test whether $SIM_N(q)$ is greater than an user-defined threshold t. The threshold t can be set using Chebyshev's inequality. This is our proposed solution for P2.

Once again, notice that the PST construction and query time can be reduced by choosing a smaller L. This is possible (without loss of accuracy) if the objective is to solely mine for outliers.

2 Probabilistic Suffix Tree

The Probabilistic Suffix Tree (PST) is a compact representation of a variable-order markov chain, which uses a suffix tree as its index structure. It was originally proposed together with the Probabilistic Suffix Automata (PSA) by Ron et. al [12]. A PST is considered to have a more memory efficient representation than the PSA. Since then, it has been used in several domains as an efficient approach for classifying sequences [4, 5, 13, 14, 15].

Figure 1 shows an example of a PST of a sequence database over the alphabet $\Sigma = \{a, b\}$. In this example each node of a PST has at most two (the size of the alphabet) children. Each edge is labelled by a symbol of the alphabet and each node is labelled by a string, which represents a path from the node to the root.

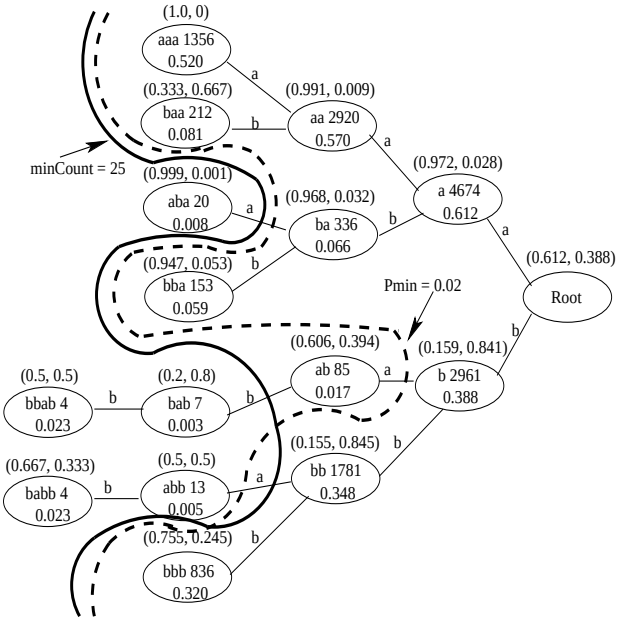

Figure 1: An example of PST and pruning it using $MinCount$ or $Pmin$. The probability distribution vectors are shown on the top of the nodes, and the label strings, the number of times they appear in the dataset and their empirical probability are shown within the nodes

The node also records a probability distribution vector of the symbols, which corresponds to the conditional probabilities of seeing a symbol right after the label string in the dataset. For example, the probability vector for the node labelled *bba* is (0.947,0.053). This means the conditional probability of seeing a right after *bba* ($P(a|bba)$) is 0.947, and seeing b right after *bba* ($P(b|bba)$) is 0.053.

The structure of PST is similar to the classical suffix tree (ST). However, there are some important differences. Besides keeping a probability distribution vector at each node, in a PST, the parent of a node is a suffix of the node, while in a classical ST the parent of a node is a prefix of the node.

2.1 Pruning of a PST

The size of a PST is a function of the cardinality of the alphabet ($|\Sigma|$) and maximum memory length L. A fully grown unchecked PST is ($O(|\Sigma|^L)$). Several pruning mechanisms have to be employed to control the size of the PST.

Bejerano and Yona [5] have proposed a two-step mechanism to prune a PST. In the first step, an empirical probability threshold $Pmin$ is used to decide whether to extend a child node. For example, at the node labelled bb, if $P(abb) \geq Pmin$, the node with label string abb will be added to the PST under some conditions. Otherwise, the node itself, including all its descendants will be ignored. The formula of computing $P(abb)$ is listed in Table 1

In the second step, a tree depth threshold L is employed to cut the PST. This means when the length of the label string of a node reaches L, its children nodes will be pruned.

Instead of using $Pmin$, Yang and Wang [15] suggested the use of $minCount$ for pruning a PST. For each node, the number of times its label string appears in the database is counted. If this number is smaller than $minCount$, then the node (and therefore all its children) are pruned.

In Figure 1 both $Pmin$ and $MinCount$ are shown in each node for ease of exposition. However, it is not necessary to keep them in the PST. The dashed and the solid lines show examples of pruning the PST using $Pmin = 0.02$ and $MinCount = 25$ respectively.

2.2 Computing Probabilities Using a PST

The probability associated with a sequence s over a PST is $P^T(s) = P^T(s_1)P^T(s_2|s_1)\dots P^T(s_l|s_1s_2...s_{l-1})$. The PST allows an efficient computation of these intermediate conditional probability terms.

For example let us compute $P^T(b|abab)$ from the PST in Figure 1. The search starts from the root and traverse along the path $\rightarrow b \rightarrow a \rightarrow b$, which is in the reverse order of string *abab*. The search stops at the node with label *bab*, because this is the longest suffix of *abab* that can be found in the PST, and $P^T(b|abab)$ is estimated by $P^T(b|bab) = 0.8$. Thus, we are exploiting the *short memory* feature, which occurs in sequences generated from natural sources: the empirical probability distribution of the next symbol, given the preceding subsequence, can be approximated by observing no more than the last L symbols in that subsequence [12, 5].

If the PST is pruned using $minCount = 25$, the search stops at the node with label *ab* and $P^T(b|abab)$ is estimated by $P^T(b|ab) = 0.394$. The following is an example to compute the probability of string *ababb* over the PST pruned using $minCount = 25$.

$$\begin{aligned} P^T(S) &= P^T(a)P^T(b|a)P^T(a|ab)P^T(b|aba)P^T(b|abab) \\ &= 0.612 \times 0.028 \times 0.606 \times 0.032 \times 0.394 \\ &= 1.309 * 10^{-4} \end{aligned}$$

Since the probabilities are multiplied, care must be taken to avoid the presence of zero probability. Thus, a smoothing procedure is employed across each node of the PST and the probability distribution vector is

perturbed to make all its components non-zero. For example, at the node with label aba, the original values of $P(a|aba)$ and $P(b|aba)$ are 1.0 and 0. This means that the symbol b is never observed right after aba. A minimum probability value, in this case 0.001, is assigned to $P(b|aba)$ and $P(a|aba)$ is adjusted to 0.999.

3 Algorithm

The PST construction algorithm is shown in Table 2. The main innovation of our approach is to use the hash map data structure at each level to efficiently retrieve and update the counts of each character before and after a segment in the sequential database. For example, suppose we are at level 2 and the alphabet is $\{a, b\}$. Then, without pruning, the hash.keys at level 2 are all the possible 2 combinations of the alphabet:$\{aa, ab, ba, bb\}$. These combinations are lexicographically ordered and the orders are stored as the values of the hash map. Thus, hash.values are $\{0, 1, 2, 3\}$. Now, the hash.key, hash.value combination is used as an index to the arrays A_{before} and A_{after}. The size of these arrays is the size of the alphabet, and the value of each element of A_{before} is the current count of $\sigma s\prime$, where $s\prime$ is the key of the hash map and σ is a character in the alphabet. Similarly A_{after} will store the count of $s\prime\sigma$. This way, in one scan, we can update all the counts at each level of the tree. After a level of the PST is constructed, the hash map is destroyed and a new hash map for the next level is initialized. For example, suppose we have a sequential database consisting of one sequence $\{abba\}$. In one scan we can update the counts of $ab \rightarrow b$, $a \leftarrow bb$, $bb \rightarrow a$ and $b \leftarrow ba$.

Theoretically, the number of entries in the hash map (i.e. the number of nodes) at level L is $|\sum|^L$ without pruning. Thus, the total complexity of this implementation is $O(NmL) + O(L|\sum|^L)$. However, if we prune the PST using $Pmin$ or $MinCount$, the number of nodes only increases exponentially at first a few levels and then decreases and converges to some constant C (see figure 7). So we can break the second part of the cost into two parts, and the total cost of constructing the PST becomes $O(NmL) + O(L|\Sigma|^\alpha) + O(LC)$, where α is fixed integer which depends upon the pruning parameters (α is usually less than 4) and C is a constant.

4 Similarity Measures

A PST provides a compact representation of a sequential database as well as an efficient mechanism to compute the factors that arise in the probability calculation. However, several measures are available to compute the similarity of a sequence with the PST. Two prominent examples are the *Odds* and the *Normalized*

The algorithm: Construct-PST($minCount$, L)

1.Initialization: let T consist of a single root node
  (with an empty label) and let k=1;
  let $S_1 \leftarrow \{\sigma | \sigma \in \sum$ and $count(\sigma) \geq MinCount\}$;
  create a hash map HM_1 for S_1 at the same time
2.While $k \leq L$
2.1 Initialize two arrays $A_{after}[|\sum|]$, $A_{before}[|\sum|]$
2.2 for each element $s\prime$ in S_k. for each symbol
2.3 $\sigma \in \sum$, A_{after} and A_{before} record the
2.4 numbers of times $s\prime\sigma$ and $\sigma s\prime$ appear in the
2.5 dataset respectively.
2.6 For each sequence s in the dataset
2.7 For each substring $s_{i,i+k-1}$ of s,
2.8 $1 \leq i \leq length(s) - k + 1$
2.9 If $s_{i,i+k-1}$ is found in HM_k then
2.10 update A_{after} and A_{before}
2.11 corresponding to $s_{i,i+k-1}$.
2.12 End If
2.13 End For
2.14 End For
2.15 For each element $s\prime$ in S_k
2.16 Add to T the node corresponding to $s\prime$
2.17 and for each $\sigma \in \sum$, compute $P(\sigma|s\prime)$
2.18 using A_{after}, smooth $P(\sigma|s\prime)$ if necessary
2.19 If there exists a symbol $\sigma\prime \in \sum$ such that
2.20 count($\sigma\prime s\prime$) $\geq MinCount$ (using A_{before})
2.21 then add $\sigma\prime s\prime$ to S_{k+1}
2.22 End If
2.23 End For
2.24 Create hash map HM_{k+1} for S_{k+1}
End While

Table 2: The PST Construction Algorithm

(by the length) measure. Given a sequence $s = s_1 \dots s_l$ and a PST T, the odds of s with respect to T is denoted as SIM_O and defined as

$$SIM_O(s, T) = \frac{P^T(s_1)P^T(s_2|s_1)\dots P^T(s_l|s_1\dots s_{l-1})}{P(s_1)P(s_2)\dots P(s_l)}$$

Thus, SIM_O represents the odds that s is a member of T (as opposed to a random sequence). Clearly, if the value of $SIM_O(s, T)$ is greater than 1, it indicates that s is more likely to be subsumed by T than be a random sequence. The SIM_O measure was used by Yang and Wang [14, 15] to decide cluster membership of a sequence s to cluster T.

The *Normalized* (by the length) measure tries to capture the length of a sequence in the similarity

computation. It is denoted as SIM_N and defined as

$$SIM_N(s,T) = \frac{1}{l}(logP^T(s_1) + \sum_{j=2}^{l} logP^T(s_j|s_1 \dots s_{j-1}))$$

The SIM_N measure explicitly captures the length of sequence in the similarity calculation. While [14, 15] mention the use of normalization in computing the similarity, all their experiments are based on the SIM_O measure. Our experiments in the next section will clearly demonstrate that SIM_N is superior when it comes to outlier detection.

In our experiments we have used the log of SIM_O because otherwise SIM_O can assume very large values.

5 Experiments, Results and Analysis

We have carried out extensive experiments to validate our approach to detect outliers in sequential databases. The results of six sets of experiments are presented in this section.

1. The first experiment was designed to test whether SIM_N and SIM_O are well normalized with respect to the length.

2. The second set of experiments was used to test the hypothesis that sequences with low similarity value (SIM_N) are found closer to the root of the PST.

3. The goal of the third experiment was to determine which measure of pruning $Pmin$ or $MinCount$ was more suitable for outlier detection.

4. In the fourth experiment, we compared the running time of our implementation for constructing a PST against Bejerano's implementation [3].

5. In the fifth set of experiments, we have compared and contrasted the two measures SIM_O and SIM_N to determine which one of the two is more suitable for outlier detection.

6. The objective of the sixth experiment was to determine if removal of sequences with low/high similarity values decrease/increase the entropy of the remaining set.

5.1 The Dataset Used

The dataset we used was downloaded from the Pfam (Protein Families database of alignments and HMMs) website. We use the curated part of Pfam which contains over 900,000 protein sequences belonging to 7868 protein families. Depending upon the experiment, either the whole data set or data belonging to some selected families was used. We have also used a synthetic data set to determine how the size of the PST nodes grows at each level. We have created two synthetic data sets, Syn1 and Syn2 on an alphabet $\{a, b\}$. In Syn1 each a and b are equally likely to appear in a sequence (1:1). In Syn2, the generator is biased towards emitting a over b (3:1).

5.2 The relationship between the length of sequence and SIM_N/SIM_O

The sequences in the dataset have different lengths. The similarity measures proposed for detecting outliers must get rid of the effect of length. In order to test whether SIM_N and SIM_O are well normalized on the length of sequences, we constructed the probabilistic suffix tree on the whole Pfam dataset, computed the similarity values for each sequences using SIM_N and SIM_O respectively, and then calculated the average similarity value for each length, which have more than 100 protein sequences.

Figure 2 and 3 show the relationship between the average value and length for SIM_N and SIM_O respectively. Both measures catch the same intrinsic characteristics of the dataset, because the peak values always appear at the same positions. However, the average similarity value for measure SIM_O goes up as length increases, while no such trend is apparent for SIM_N. So for the purpose of outlier detection, the measure SIM_N is better than SIM_O, because the later one is likely to pick up short sequences as outliers.

5.3 The Tree Depth

The size of a PST is an exponential function of the length of the short memory, L. This make it mandatory to manage the growth of the PST. One of our hypotheses is that outliers can be mined by just using information in the top part of a PST. The experiment results confirm this hypothesis.

Figure 4 shows the relationship between the similarity values and the maximum depth we need to search in a PST while computing these values. Even though it is not necessary to go deep inside a PST to compute high similarity values, low similarity values (i.e., candidate outliers) can be computed by traversing nodes closer to the root. More than half of the points in Figure 4 stay below the depth (closer to the root) of twenty five in the tree.

All the protein sequences in the $Pfam$ database belong to 7868 different families. In general sequences that belong to one family are structurally similar and should have high degree of "similarity" with each other. Thus, sequences in one family should be outliers with respect to another family - and a good similarity measure should be able to pick this up. There are a few exceptions, as some proteins have complex structures and may be sim-