Unsupervised Sequence Segmentation by a Mixture of Switching Variable Memory Markov Sources

Yevgeny Seldin SELDIN@CS.HUJI.AC.IL
Gill Bejerano JILL@CS.HUJI.AC.IL
Naftali Tishby TISHBY@CS.HUJI.AC.IL
School of Computer Science and Engineering, The Hebrew University, Jerusalem 91904, Israel

Abstract

We present a novel information theoretic algorithm for unsupervised segmentation of sequences into alternating Variable Memory Markov sources. The algorithm is based on competitive learning between Markov models, when implemented as *Prediction Suffix Trees* (Ron et al., 1996) using the MDL principle. By applying a model clustering procedure, based on rate distortion theory combined with deterministic annealing, we obtain a hierarchical segmentation of sequences between alternating Markov sources. The algorithm seems to be self regulated and automatically avoids over segmentation. The method is applied successfully to unsupervised segmentation of multilingual texts into languages where it is able to infer correctly both the number of languages and the language switching points. When applied to protein sequence families, we demonstrate the method's ability to identify biologically meaningful sub-sequences within the proteins, which correspond to important functional sub-units called domains.

1. Introduction

Unsupervised segmentation of sequences has become a fundamental problem with many important applications such as analysis of texts, handwriting and speech, neural spike trains and bio-molecular sequences. The most common statistical approach to this problem, using hidden Markov models (HMM), was originally developed for the analysis of speech signals, but became the method of choice for statistical segmentation of most natural sequences. HMMs are predefined parametric models and their success crucially depends on the correct choice of the state model - the observation distribution attached to each of the states of the Markov chain. In the common application of HMM the architecture and topology of the model are predetermined and the memory is limited to first order. It is rather difficult to generalize these models to hierarchical structures with unknown a-priory state-topology (see (Fine et al., 1998) for an attempt).

An interesting alternative to the HMM was proposed in Ron et al. (1996) in the form of a sub class of *probabilistic finite automata*, the variable memory Markov (VMM) sources. While these models can be weaker as generative models, they have several important advantages: (i) they capture longer correlations and higher order statistics of the sequence; (ii) they can be learned in a provably optimal PAC like sense using a construction called *prediction suffix tree (PST)*(Ron et al., 1996); (iii) they can be learned very efficiently by linear time algorithms (Apostolico & Bejerano, 2000); and (iv) their topology and complexity are determined by the data.

This paper presents a powerful new extension of the VMM model and the PST algorithm to a stochastic mixture of such models, that are learned in a hierarchical competitive way using a deterministic annealing (DA) (Rose, 1998) approach. This problem is generally computationally hard, similarly to data clustering. Only very simple sequences can be correctly segmented efficiently in general (Freund & Ron, 1995). Our model can in fact be viewed as an HMM with a VMM attached to each state, but the learning algorithm allows a completely adaptive structure and topology both for each state and for the whole model. The approach we take is information theoretic in nature. The goal is to enable short description of the data by a (soft) mixture of variable memory Markov models, each one controlled by an MDL principle (see (Barron et al., 1998) for a review). This we do by modifying the original PST algorithm using the MDL formulation, while preserving its good learn-ability properties. The mixture model is then learned via a generalized *rate distortion theory* (see Cover and Thomas (1991), Ch. 13) approach. Here we take the *log-likelihood* of the data by each model as an effective *distortion measure* between the sequence and its representative model and apply the Blahut-Arimoto (BA) algorithm (see Cover and Thomas (1991)) to optimally partition the sequence(s) between the VMM model centroids. Just like in many clustering algorithms we then update the models based on this optimal partition of the sequence(s). In this way a natural resolution parameter is introduced through the constraint on the expected tolerated distortion. This "temperature" like Lagrange multiplier is further used in the deterministic annealing loop to control the resolution of the model. The hierarchical structure is obtained by allowing the models to split (the *refinement* step) after convergence of the iterations between the BA algorithm and the VMM centroids update.

This new algorithm exhibits several interesting features which will be further discussed elsewhere. It turns out that the interplay between the MDL and the DA procedure prevents "over segmentation", by eliminating small models that fails to capture enough data. The model is thus "self regulated" in an interesting way. The algorithm is described in Sec. 2, 3 and further discussed in Sec. 5.

In Sec. 4 we apply the algorithm to two types of datasets. The first is a mixture of interchanged texts in 5 different European languages. The model was able to identify both the correct number of languages and the segmentation of the text sequence between the languages to a resolution of a few letters. We then apply the algorithm to the much harder problem of protein segmentation. We briefly show here that the algorithm is able to identify biologically meaningful sub-sequences within the proteins, which correspond to important functional sub-units known as protein domains. This extends earlier work on protein classification using the PST algorithm (Bejerano & Yona, 2001) and opens a way for new applications of this approach in bioinformatics, further pursued in (Bejerano et al., 2001) and elsewhere.

2. Single Source Modeling

In this section we will define variable memory Markov processes, review an efficient data structure for their representation from (Ron et al., 1996) and present a new non-parametric learning algorithm that we will later use as a core for the segmentation process.

2.1 Variable Memory Markov Processes

Given a string $\bar{x}$, over a finite alphabet Σ, that was sequentially generated by some statistical source G, the probability that G has generated that particular sequence can always be written as: $P_G(\bar{x}) = P_G(x_1..x_n) = \prod_{i=1}^{n} P_G(x_i|x_1..x_{i-1})$. In this section we assume G to be stationary and ergodic (Cover & Thomas, 1991). We define a *context* of x_i to be any substring $x_{i-m}..x_{i-1}$ for $m \geq 0$. If $m = 0$ we say that the context of x_i is the empty string, denoted by λ. Further we define C to be any finite subset of strings in Σ^* that includes λ. We say that $x_{i-m}..x_{i-1}$, or λ, is the *C-context* of x_i if it is the longest suffix of $x_1..x_{i-1}$ in C. Process G respects context set C if $P_G(x_i|x_1..x_{i-1}) = P_G(x_i|C\text{-}context(x_i))$ for all i. The length of C-context(x_i) is the memory of process G at place i, and it may vary with i.

2.2 Prediction Suffix Trees (PSTs)

A context set C may be efficiently represented using a tree. By associating a distribution vector over Σ with each node of the tree we get a PST[1] (see Fig. 1). Formally, a PST T is a $|\Sigma|$-ary tree that satisfies:

1. For each node each outgoing edge is labeled by a single symbol $\sigma \in \Sigma$, while there is at most one edge labeled by each symbol.
2. Each node of the tree is labeled by a *unique* string s (a context) that corresponds to a 'walk' starting from that node and ending in the root of the tree. We identify nodes with their labels and label the root node by the empty string λ.
3. A probability vector $P_s(\sigma)$ is associated with each node s. $P_s(\sigma)$ represents the distribution over the symbol coming immediately after context s[2].

We define $suf_T(x_1..x_i)$ as the longest sequence $x_{i-m}..x_i$ that makes a path in T when we start from the root and traverse the edge labeled by x_i, from there we traverse the edge labeled by x_{i-1} etc., until there is no appropriate edge to continue with or we have traversed the whole string[3]. If there is no edge labeled by x_i at the root we say that $suf_T(x_1..x_i) = \lambda$. The collection of all node labels in T make up our set of memorized contexts. (It is easy to see that any context set may be represented by a PST.)

[1] A Prediction Suffix Tree is related to, but differs from a classical suffix tree (see (Apostolico & Bejerano, 2000)).

[2] $P_s(\sigma) = P(\text{next symbol is } \sigma | \text{last symbols were } s)$.

[3] Note that we do not necessarily stop at a leaf.

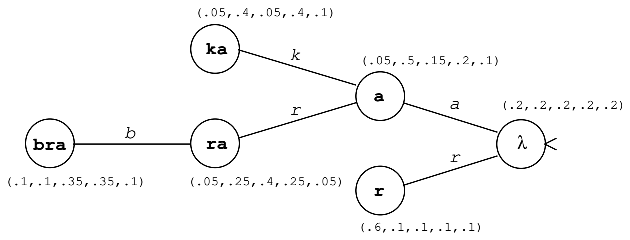

Figure 1. **An example of a PST** over the alphabet $\Sigma = \{a, b, k, l, r\}$. The vector near each node is the probability distribution for the next symbol. E.g., the probability to observe k after the substring $bara$, whose largest suffix in the tree is ra, is $P(k|bara) = P_{ra}(k) = 0.4$.

2.3 Predicting and Generating using PSTs

Here we define the probability measure that a PST T induces on the space of all strings $\bar{x} \in \Sigma^n$, for any given n. Given a string $\bar{x} \in \Sigma^n$ and a PST T the probability that $\bar{x}$ was generated by T is:

$$P_T(\bar{x}) = \prod_{i=1}^{n} P_T(x_i|x_1..x_{i-1}) = \prod_{i=1}^{n} P_{suf_T(x_1..x_{i-1})}(x_i)$$

When T is used as a generator, it generates a symbol x_i according to the distribution $P_{suf_T(x_1..x_{i-1})}$.

For the sake of consistency we would like the internal nodes of T to hold marginal distributions: $P_s(\sigma) = \sum_{\hat{\sigma}\in\Sigma} \frac{P_T(\hat{\sigma}s)}{P_T(s)} P_{\hat{\sigma}s}(\sigma)$.

2.4 Learning PSTs

We now turn to present a new MDL driven algorithm for PST learning. The algorithm is non-parametric and exhibits self-regularization. It is generalized to handle weighted data, which will appear later on.

The inputs to the algorithm are a string $\bar{x} = x_1..x_n$ and a vector of weights $\bar{w} = w_1..w_n$, where each w_i is a weight associated with x_i $(0 \le w_i \le 1)$[4]. We will denote $w(x_i) \equiv w_i$. You may think of $w(x_i)$ as a measure of confidence we give to the observation x_i. For now you may assume all $w_i = 1$.

For a string s we say that $sx_i \in \bar{x}$ if it is a substring of $\bar{x}$ ending at place i. We define:

$$w_s(\sigma) \equiv \sum_{x_i=\sigma \ and \ sx_i\in\bar{x}} w(x_i)$$

and $w(s) \equiv \sum_{\sigma\in\Sigma} w_s(\sigma)$. Clearly $\frac{w_s(\sigma)}{w(s)}$ is an empirical estimate for $P_s(\sigma)$.

[4]Generalization to a set of strings is straightforward and therefore omitted here for ease of notation.

The idea behind MDL is to minimize the total length (in bits) of model description together with the code length of the data when it is encoded using the model. When coding a single node s we should enumerate its sons and encode the distribution vector P_s. The first takes $|\Sigma|$ bits - bit σ denotes the presence of son σ. For the second it is sufficient to code all the counts $w_s(\sigma)$. Since the total amount of data "passing through" node s[5] is $w(s)$ the counts should be coded to within accuracy $\sqrt{w(s)}$. Thus the description size of s is:

$$Size(s) = |\Sigma| + \frac{|\Sigma|}{2} \cdot \log_2(w(s))$$

Denoting by T_s the subtree of T rooted at node s:

$$Size(T_s) = Size(s) + \sum_{\sigma s\in T} Size(T_{\sigma s})$$

($s \in T$ means that s is a node in T). The minimal average code length per symbol, for all symbols coded using node s, is given by the *entropy* of P_s, $H(P_s) \equiv -\sum_{\sigma\in\Sigma} P_s(\sigma) \cdot \log_2(P_s(\sigma))$. The equivalent quantity for a subtree T_s is thus a weighted sum given by:

$$H(T_s) = \sum_{\sigma s\in T} \frac{w(\sigma s)}{w(s)} \cdot H(T_{\sigma s}) + \sum_{\sigma s\notin T} \frac{w(\sigma s)}{w(s)} \cdot H(P_s)$$

Summing this altogether we get:

$$TotalSize(T_s) = Size(T_s) + w(s) \cdot H(T_s)$$

Our goal is to minimize $TotalSize(\lambda)$ which is the total description length of the whole tree together with all coded data (as all data passes through the root node λ). The algorithm works in two steps. In step I we extend all the nodes that are potentially beneficial, i.e. by using them we *may* decrease the total size. Clearly only those nodes whose description size is smaller than the code length of data passing through them when that data is coded using the parent node distribution are of interest. In step II the tree is recursively pruned so that only truly beneficial nodes remain. If a child subtree $T_{\sigma s}$ of some node s gives better compression (respecting its own description length) than that of its parent node, that subtree is left, otherwise it is pruned. The algorithm is given in Fig. 2

3. Sequence Segmentation Algorithm

Now suppose that a given string $\bar{x}$ was generated by repeatedly switching between several different PST models with some upper bound on the alternation rate. I.e., there are k PSTs and a partition of $\bar{x}$ into

[5]$suf(x_1..x_{i-1})$ ends with s.

Learn_PST(String $\bar{x}$, Weights $\bar{w}$)

1. T = Build_PST($\bar{x}$, $\bar{w}$)
2. Prune(T, λ)

The two steps:

I. Build_PST (String $\bar{x}$, Weights $\bar{w}$)

1. Start with T having a single node λ.
2. Recursively for each $s \in T$ and $\sigma \in \Sigma$
   If $Size(\sigma s) < H(P_s) \cdot w(\sigma s)$ Then
   Add node σs to T.

II. Prune (Tree T, node s)

1. For each $\sigma \in \Sigma$ such that $\sigma s \in T$:
   (a) Prune(σs)
   (b) If $TotalSize(T_{\sigma s}) > H(P_s) \cdot w(\sigma s)$ Then
   Delete subtree $T_{\sigma s}$

Figure 2. The PST learning algorithm.

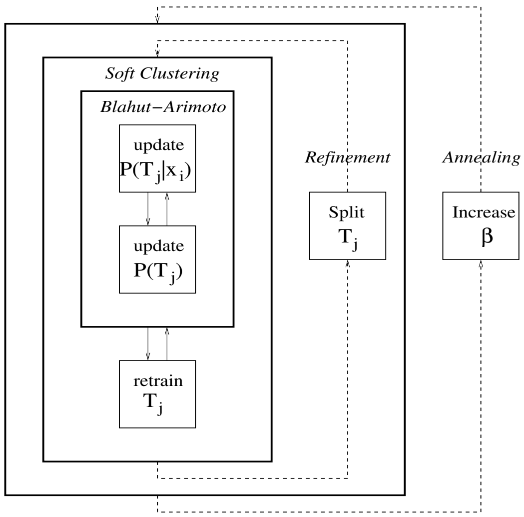

Figure 3. **A schematic description of the algorithm.**

$l \geq k$ contiguous segments, with length of each segment greater than some constant value L, such that each segment was generated by a single PST out of k. Our goal is to find k' PST models and a segmentation of $\bar{x}$ that will be as close as possible to the original ones.

This problem is similar to the problem of finding the best number and parameters for a Gaussian mixture model of points in R^n. Given a string $\bar{x}$ and a vector of assignment probabilities we can build a PST model and estimate its parameters. Alternatively, a given model induces probabilities on all substrings of $\bar{x}$. Alternating between these two estimations is the essence of the EM algorithm in any mixture model. This alternating estimation algorithm can be embedded in a deterministic annealing (DA) procedure to allow for increasing resolution, or number of mixture components. In our case, however, we do not allow our PST models to switch at every symbol, but rather require contiguous segments. The fundamental reason for limiting the model switching frequency is that too short segments do not enable reliable discrimination between different models.

We apply deterministic annealing since it can avoid local minima effectively and it is an elegant framework for generating hierarchical structures, though it may produce sub-optimal results (see (Rose, 1998)).

Next we give some definitions and describe the Blahut-Arimoto and our soft clustering algorithm. We then embed it in the DA framework to obtain the hierarchical segmentation. See Fig. 3 for schematic description of the complete algorithm.

3.1 Definitions

Let $\mathcal{T} = \{T_j\}_{j=1}^{k}$ be the set of PSTs of size k we are currently working with. We define $w_j(x_i) \equiv P(T_j|x_i)$ to be the probability that a symbol x_i is assigned to model T_j[6].

In order to estimate the quality of a given partition we define a distance (local distortion) between a symbol x_i and a model T_j to be negative log likelihood of T_j on a window of size $2M+1$ around x_i:

$$d(x_i, T_j) = -\sum_{\alpha=i-M}^{i+M} \ln P_{T_j}(x_\alpha | x_1..x_{\alpha-1}) \ .$$

The role of the window is to smooth the segmentation and to enable reliable estimation of the log-likelihood. The *global distortion*, i.e. the average distance between segments and the corresponding models, of an assignment is given by:

$$\langle d \rangle = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} d(x_i, T_j) \cdot P(T_j|x_i) \ .$$

3.2 The Blahut-Arimoto Algorithm

First we want to find the optimal assignment probabilities $P(T_j|x_i)$ for a *fixed* set of PST models, $\mathcal{T}$, constrained by the allowed distortion level D. Rate distortion theory (Cover & Thomas, 1991, Ch. 13) provides us with the optimal assignment via:

$$\min_{\{P(T_j|x_i)\ :\ \langle d \rangle \leq D, \sum_{j=1}^{k} P(T_j|x_i)=1\}} I(\bar{x}, \mathcal{T}) \qquad (1)$$

[6]The vector of weights $\bar{w}_j$ is later used to retrain T_j.

where I is the *mutual information* between $\bar{x}$ and $\mathcal{T}$

$$I(\bar{x}, \mathcal{T}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} P(T_j|x_i) \cdot \log \frac{P(T_j|x_i)}{P(T_j)}$$

and $P(T_j)$ the proportion of data assigned to model j

$$P(T_j) = \frac{1}{n} \sum_{i=1}^{n} P(T_j|x_i)$$

In rate distortion theory Eq. 1 is called the *rate distortion function*, and is denoted by $R(D)$.

By minimizing the mutual information we in fact enable minimal description length of the sequences using the PST models, subject to a given distortion constraint. Since our distortion, an expected log-likelihood, is also the optimal code length by the model, it is fully consistent with the MDL framework. We thus try to find a mixture of PSTs that enable short description of the complete observation sequence, under some contiguity requirements from the resulting segmentation.

We employ the alternating minimization procedure, known as the Blahut-Arimoto algorithm, which is guaranteed to converge to the optimal assignment:

Blahut-Arimoto($P(T_1), .., P(T_k)$, β)

Repeat until convergence:

1. $\forall i, j: \;\; P(T_j|x_i) = \frac{P(T_j)e^{-\beta d(x_i, T_j)}}{\sum_{\alpha=1}^{k} P(T_\alpha)e^{-\beta d(x_i, T_\alpha)}}$
2. $\forall j: \;\; P(T_j) = \frac{1}{n} \sum_{i=1}^{n} P(T_j|x_i)$

Here the distortion constraint, D, is imposed by the corresponding Lagrange multiplier β.

3.3 Soft Clustering

Now we go one step further by allowing to modify the PST models. This is analogous to the centroid re-estimation in clustering. We want to obtain a good (low distortion) segmentation of $\bar{x}$ for a given value of β (the assignment probabilities are given by 1. in the BA algorithm).

We approach this problem using a soft clustering procedure. Given an initial set of k PSTs $\mathcal{T}$, we partition the sequence using the BA algorithm and then retrain all k PSTs, using the assignment probabilities $P(T_j|x_i)$ obtained from the BA as weight vectors $\bar{w}_j$ for the Learn_PST procedure. These two steps are repeated until convergence:

Soft_Clustering($\mathcal{T}$, $P(T_1), .., P(T_k)$, β)

Repeat until convergence:

1. Blahut-Arimoto($P(T_1), .., P(T_k)$, β)
2. $\forall j: \; T_j =$ Learn_PST($\bar{x}$, $\bar{w}_j$)

Here the Lagrange multiplier β plays the role of *resolution* parameter and prevents from falling into local minima.

At every given distortion level, D, a limited number of PSTs K is sufficient to achieve D. When $k > K$ some of the PSTs collapse into a single model - a phenomenon clearly described in (Rose, 1998) - or remain without data ($P(T_j) = 0$). The latter is caused by the requirement of having contiguous segments in the final segmentation. Because of this requirement the competition between the models "pushes out" the models who don't "acquire" enough data in favor of those having more data. In this manner the algorithm "self regulates" its global complexity.

3.4 Deterministic Annealing and the Segmentation Algorithm

The landscape of the problem defined in this section is typically riddled with local minima and it is computationally difficult to obtain the optimal solution. Usually a successful way of finding a good solution is through deterministic annealing: a *series* of solutions to the soft clustering problem is found, starting from a low value of *resolution* (inverse "temperature") parameter β and gradually increasing it, while allowing models to split in two when necessary.

The splitting procedure is straightforward:

Split_PSTs($\mathcal{T}$, $P(T_1), .., P(T_k)$)

Replace each T_j in $\mathcal{T}$ by two new models:

1. Start with two exact copies of T_j: T_{j_1} and T_{j_2}
2. For each node s in T_j and for each $\sigma \in \Sigma$:
   (a) Select $\{\zeta = 1, \xi = 2\}$ or $\{\zeta = 2, \xi = 1\}$ with probability $\frac{1}{2}$ / $\frac{1}{2}$.
   (b) Perturb and normalize the counts vectors:
   For T_{j_ζ}: $w_s(\sigma) = (1 + \gamma) \cdot w_s(\sigma) \quad (|\gamma| \ll 1)$
   For T_{j_ξ}: $w_s(\sigma) = (1 - \gamma) \cdot w_s(\sigma)$
3. $P(T_{j_1}) = \frac{1}{2}P(T_j)$, $P(T_{j_2}) = \frac{1}{2}P(T_j)$

For each PST T in $\mathcal{T}$ we create two copies of T and perform random antisymmetric perturbations of the counts vectors in each node of the two copies. Then we *replace* T with the two obtained PSTs while distributing $P(T)$ equally among them.

We are finally ready to outline the complete algorithm.

We start with $\mathcal{T}$ including a single "average" PST T that is trained on the full sequence $\bar{x}$ with $w(x_i) = 1$ for all i. We pick an initial value of β, split $\mathcal{T}$ and partition $\bar{x}$ among the resulting models, T_1 and T_2. We then split $\mathcal{T}$ again and repeat. If a model is found to have lost its data it is eliminated. When the number of survived models stops increasing we increase β and then repeat the whole process.

The segmentation algorithm:

Initialization:

For all i, $w_1(x_i) = 1$
$T_1 = \text{Learn\_PST}(\bar{x}, \bar{w}_1)$
$\mathcal{T} = \{T_1\}$, $P(T_1) = 1$
$\beta = \beta_0$, $k_{prev} = |\mathcal{T}|$

Annealing loop:

1. Split_PSTs($\mathcal{T}$, $P(T_1), .., P(T_k)$)
2. Soft_Clustering($\mathcal{T}$, $P(T_1), .., P(T_k)$, β)
3. Remove all T_j such that $P(T_j) = 0$ from $\mathcal{T}$.
4. If $k_{prev} \geq |\mathcal{T}|$ then
   Increase β
5. $k_{prev} = |\mathcal{T}|$

Sets of segments that are assigned with high probability to the same model over a long range of β are *stable clusters* that contain important information about the statistical structure of our sample.

4. Applications: Multilingual Text and Protein Sequence Segmentation

In our first example we construct a synthetic text composed of alternating fragments of five other texts in five different languages: English, German, Italian, French and Russian, using standard transcripts to convert all into lower case Latin letters with blank substituting all separators. The length of each fragment taken is 100 letters, which means that we are switching languages every two sentences or so. The total length of the text was 150000 letters (30000 from each language).

We made several independent runs of our algorithm. In every run, after 2000-3000 accumulated innermost (BA) iterations we got a clear-cut, correct segmentation of the text into segments corresponding to the different languages, accurate up to a few letters (See Fig. 4, 5 for a typical example)[7]. Moreover, in all runs, further splitting of all 5 language models resulted in starvation and subsequent removal of 5 extra models, taking us back to the same segmentation as before.

[7]Correct segmentation was achieved even at a switching rate of 50 letters per segment, but of poorer quality.

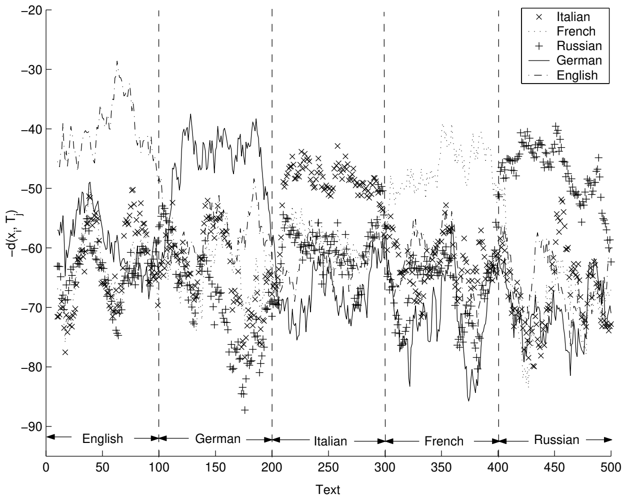

Figure 4. **Multilingual text segmentation**. The graph shows $-d(x_i, T_j)$ for the first 500 letters of text, for all $T_j \in \{T_{English}, T_{German}, T_{Italian}, T_{French}, T_{Russian}\}$. The true segmentation appears at the bottom of the graph.

Also, in most runs linguistically similar languages (English and German, French and Italian) separated at later stages of the segmentation process (Fig. 6 gives an example), suggesting a hierarchical structure over the discovered data sources.

Next, we briefly demonstrate the potential of applying our method to protein domain discovery and classification. A domain is defined as an autonomous *functional* sub-unit of a protein. Having used a family of related protein sequences as our text[8], Fig. 7 shows for a typical family member how our models have pinpointed the two known domains of this protein. The domains are seen here to coincide with the two conserved regions elucidated in this protein family. The ability to segment, and in fact sub-classify these domains in an automated unsupervised manner holds great potential and is further explored in (Bejerano et al., 2001).

Following a referee's suggestion we tried limiting the depth (and thus power) of our PST models to examine their added benefit. For languages, using zero depth PSTs (single root node) deteriorated the performance drastically. The best we could separate were two languages alternating every 200 letters, and even that at very poor quality. However, already when limited to depth one, the models performed comparable to the unrestricted models both for languages and proteins (with some loss in signal strength). This attests to the fact that first order dependencies already suffice to dif-

[8]Each protein sequence is represented by a string over a standard alphabet of the 20 amino acids.

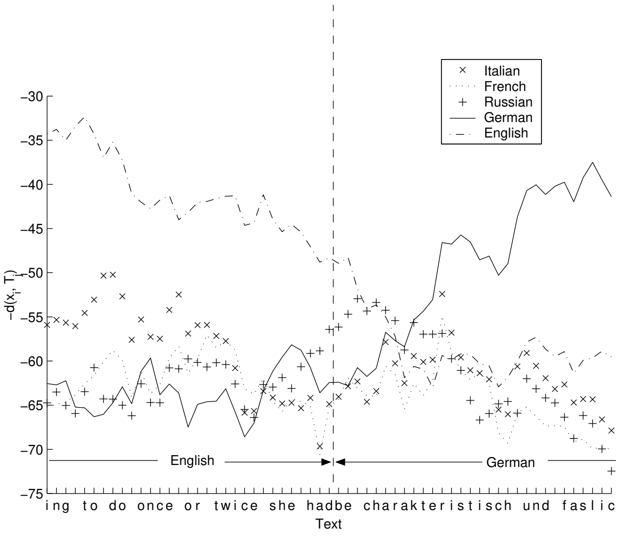

Figure 5. **Zoom in on a transition region from English to German**. The text on the x-axis corresponds to $i = 70..130$. Note the correspondence of the transition point into German to the English-like beginning "*be chara*kteristisch..." of the German segment.

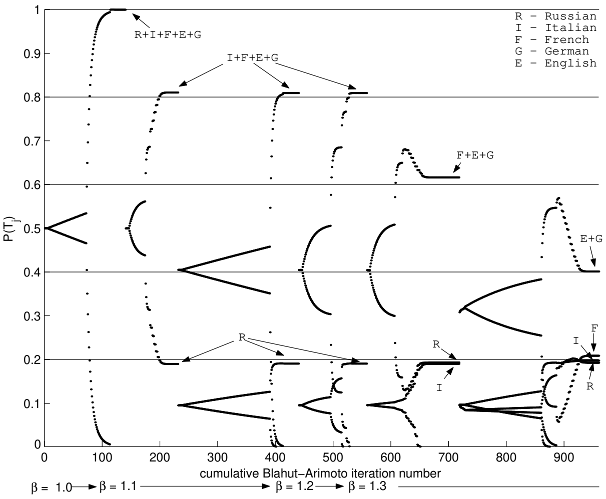

Figure 6. **Source separation**. The proportion of text assigned to each model is shown for all models, as a function of the iterations of our innermost loop. Bifurcations are the splitting events and graphs dropping off to zero show models dying out. Increments in β occur after the number of models converges at a given temperature. Languages captured by each model after the soft clustering has converged are pointed out. Notice how the order in which the languages separate from the primary joint model matches language relatedness.

ferentiate between sources in these particular cases.

We have also tried to restrict the BA loop to a single pass and it appears that for some cases this improves the performance. This happens because fewer BA iterations leave more place for the soft clustering (and thus model retraining) steps, and by looking through a wider range of (intermediate) models the system is able to "sense" and attain better solutions.

5. Discussion and Further Work

The sequence segmentation algorithm we describe and evaluate in this paper is a combination of several different information theoretic ideas and principles, naturally combined into one new coherent procedure. The core algorithm, the construction of Prediction Suffix Trees, is essentially a source coding *loss-less compression* method. It approximates a complex stochastic sequence by a probabilistic automaton, or a Markov model with variable memory length. The power of this procedure, as demonstrated on both natural texts and on protein sequences, is in its ability to capture short strings (suffixes) that are significant predictors - thus good features - for the statistical source. We combine the PST construction with another information theoretic idea - the MDL principle - and obtain a more efficient estimation of the PST, compared with its original learning algorithm.

Our second key idea is to embed the PST construction in a *lossy compression* framework by adopting the rate-distortion theory into a competitive learning procedure. Here we treat the PST as a model of a single statistical source and use the rate distortion framework (i.e., the Blahut-Arimoto algorithm) to partition the sequences between several such models in an optimal way. Here we specifically obtain a more expressive statistical model, as *mixtures* of (short memory, ergodic) Markov models lay outside of this class, and can be captured only by much deeper Markov models. This is a clear advantage of our current approach over mixtures of HMMs (as done in (Fine et al., 1998)) since mixtures of HMMs are just HMMs with constrained state topology.

The analogy with rate-distortion theory enables us to take advantage of the trade-off between compression (rate) and distortion, and use the Lagrange multiplier β, required to implement this trade-off, as a resolution parameter. The deterministic annealing framework follows naturally in this formulation and provides us with a simple way to obtain hierarchical segmentation of very complex sequences. As long as the underlying statistical sources are distinct enough, compared to the average alternation rate between them, our segmentation scheme should perform well.

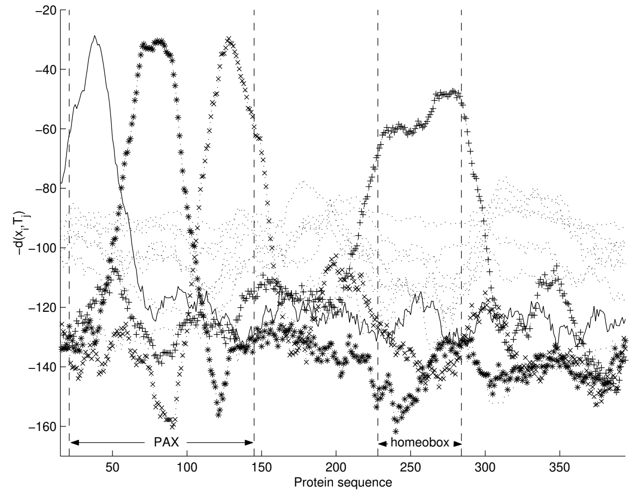

Figure 7. **Segmentation of a member of the Paired Box protein family**. The PAX6 SS protein (Swissprot acc. O57582) is shown. The boundaries of the two functional domains of the protein are marked, showing excellent fit with our unsupervised segmentation, which in fact suggests a finer sub division within the first domain.

Several natural extensions of our ideas are possible. One may replace the PST with even more efficient loss-less coding schemes, such as the Context-Tree-Weighting algorithm (Willems et al., 1995). This powerful method trades the MDL principle for a Bayesian formulation, essentially averaging efficiently over all possible PSTs for a given sequence. The disadvantages are the much larger data structures required here, and the absence of the clear features that emerge from the VMM model. An opposite approach, hinted toward the end of Sec. 4 suggests that we may benefit from replacing our generative source models with discriminative ones, aimed not so much at capturing the statistical richness in each source but rather at differentiating between them.

Another interesting extension is the segmentation of one sequence based on the prediction of another, related, sequence (such as the prediction of protein structure from its sequence) by replacing the PST with its kin, the *probabilistic transducer* (Singer, 1997). This can then be embedded in the *information bottleneck method* (Tishby et al., 1999), which extends rate distortion theory to relevant compression of one sequence with respect to another one, to enable relevant segmentation of related *pairs* of natural sequences.

Acknowledgments

The authors wish to thank Hanah Margalit for useful discussions. GB is supported by a grant from the Ministry of Science, Israel. Work is partly supported by grants from the US-Israel Bi-national Science Foundation (BSF) and the German Israeli Foundation (GIF).

References

Apostolico, A., & Bejerano, G. (2000). Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *J. Comput. Biol.*, *7*, 381–393.

Barron, A., Rissanen, J., & Yu, B. (1998). The minimum description length principle in coding and modeling. *IEEE Trans. Info. Theory*, *44*, 2743–2760.

Bejerano, G., Seldin, Y., Margalit, H., & Tishby, N. (2001). Extraction of protein domains and signatures through unsupervised statistical sequence segmentation. *Preprint.*

Bejerano, G., & Yona, G. (2001). Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinform.*, *17*, 23–43.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory.* Wiley Series in Telecommunications. New York, NY: John Wiley & Sons.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical Hidden Markov Model: Analysis and applications. *Mach. Learn.*, *32*, 41–62.

Freund, Y., & Ron, D. (1995). Learning to model sequences generated by switching distributions. *Comput. Learn. Theory (COLT) 8* (pp. 41–50). New York, NY: ACM press.

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Mach. Learn.*, *25*, 117–149.

Rose, K. (1998). Deterministic annealing for clustering, compression, classification, regression and related optimization problems. *IEEE Trans. Info. Theory*, *80*, 2210–2239.

Singer, Y. (1997). Adaptive mixtures of probabilistic transducers. *Neural Comp.*, *9*, 1711–1733.

Tishby, N., Pereira, F., & Bialek, W. (1999). The information bottleneck method. In *Allerton conference on communication, control and computation*, vol. 37, 368–379.

Willems, F. M. J., Shtarkov, Y. M., & Tjalkens, T. J. (1995). The context-tree weighting method: basic properties. *IEEE Trans. Info. Theory*, *41*, 653–664.