On Unsupervised Learning of Mixtures of Markov Sources

Thesis submitted for the degree "Master of Science"

Yevgeny Seldin

School of Computer Science and Engineering
The Hebrew University of Jerusalem
2001

This work was carried out under supervision of
Prof. Naftali Tishby

To my beloved grandmothers
Rachel and Sofia,
to my grandfather
Boris
and in the memory of my grandfather
Michael.

Моим любимым бабушкам
Оке и Софе,
моему дедушке
Боре
и светлой памяти моего дедушки
Миши

Acknowledgements

First of all, I would like to thank my supervisor, prof. Naftali Tishby. I am very grateful to Tali for taking me to be his student, for teaching me many new things, and especially for opening of the great world of scientific research for me. Tali, it was a real pleasure to work with you those two years!

I would also like to thank Gill Bejerano for the help and good advices he gave me starting from my first steps in the learning lab. Gill, I highly appreciate our long and valuable discussions and your significant contribution to our joint work.

As well, I am grateful to Hanah Margalit from the department of molecular genetics and biotechnology, who did much for the biological part of our joint work.

I am also grateful to Nir Friedman and Yoram Singer for their important, both critical and constructive comments to my work throughout its development. Special gratitude to Yoram for providing a plenty of useful references related to my work and to Nir for thoroughly reading the final draft of this work, for pointing out some weak spots left and giving a number of good advices.

I want to thank the members of the machine learning lab for being just a great company, for all the warmth and help I got from them during this time.

And, certainly, I would like to thank my parents for all the care they gave me not only in those two years and for all the energy they put into me. I am also grateful to my grandparents for their great love and attention. As well, I want to thank my aunt Sveta for being a good friend and interlocutor and for keeping in me a great interest to biology in general and to molecular biology in particular. And, thanks a lot to my brother, to all my aunts and uncles, and to all my cousins and friends for just being with me - both physically and mentally.

Yevgeny Seldin

Jerusalem, Israel
December 2001

Contents

Abstract

Unsupervised classification, or clustering, is one of the basic problems in data analysis. While the problem of unsupervised classification of independent random variables has been deeply investigated, the problem of unsupervised classification of dependent random variables, and in particular the problem of segmentation of mixtures of Markov sources, has been hardly addressed. At the same time supervised classification of Markov sources has become a fundamental problem with many important applications, such as analysis of texts, handwriting and speech, neural spike trains and bio-molecular sequences. This question, previously approached with hidden Markov models (HMMs), in the last decade found additional interesting solutions using adaptive statistical models with improved learnability properties. One of such models is Prediction Suffix Tree (PST), suggested in [RST96].

Our current work comes to close the gap between our abilities in supervised and unsupervised learning. We describe and analyze a novel information theoretic algorithm for unsupervised segmentation of sequences into alternating variable memory Markov sources, first presented in [SBT01b]. The algorithm is based on a new procedure for PST learning that uses MDL principle to control PST complexity and gets no external parameters. The algorithm embeds competitive learning of the PST models into model clustering procedure, based on rate distortion theory combined with deterministic annealing. The complexity of the mixture (clustering resolution) is gradually increased through annealing of the rate-distortion tradeoff. As a result we obtain a hierarchical top-down segmentation of sequences into alternating variable memory Markov sources.

The method is successfully applied 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 (results of the [BSMT01] work), we demonstrate the method's ability to identify biologically meaningful sub-sequences within the proteins, which correspond to signatures of important functional sub-units, called domains. Our approach to proteins classification (through the obtained signatures) is shown to have both conceptual and practical advantages over the currently used methods.

Chapter 1

Introduction

Life is just a long random walk.

Devroye, Györfi, Lugosi.
A Probabilistic Theory of Pattern Recognition, 1996.

Life is just a long random walk. And being the ones walking we would naturally like to predict the future of this walk. But most times the only information we have about the future is the past we already saw and the hoped-to-be-right belief that future will behave like the past in the meaning that similar situations will result in similar development. The last assumption is based on our past experience that physical laws are time and space invariant.

Learning theory deals with the question of predicting the (results of) future events given the (results of) past events and sometimes some additional observations related to the future. The first significant results in formal definition and exploration of this question were obtained in the 1920's - 1930's by [Fis22, Fis25, Gli33, Can33, Kol33]. Though the field formed as a separate field of studies only in the 1970's - 1980's with the works of [VC71, VC81, Val84]. In general, the learning theory may be seen as an intersection of statistics with the computational theory. One may also find deep connections of the learning theory to the information theory; some of them will be discussed here.

The question of predicting the future events based on the results of the past events is known as the question of *statistical inference*. The simplest model of statistical inference is *pattern recognition* problem. Pattern recognition deals with estimation of {0,1}-valued functions. This problem is discussed in depth in [DH73, Bis95, DGL96, Vap98] and many other books. Sometimes, especially when the investigated function takes more than two, but finite number of possible values, the problem is also called *classification* (the value of the function is the index of the class its argument belongs to). A more general and hard problem of estimating real-valued functions is known as a problem of *regression estimation*, discussed in [Vap98, Bis95]. In both cases the input we get is a set of $\langle x; f(x)\rangle$ pairs where x belongs to some space we are sampling from and $f(x)$ is the value of the investigated function at x. This set of pairs is called our past, history or training sample. The "future" we want to predict is the value of f at some new point x we have not seen yet. In a slightly different formulation, x may come from some probability space $\mathcal{X}$, and f may be a probability density function over $\mathcal{X}$. Then our sample will be just a set of points sampled from $\mathcal{X}$ according to f, and we will have to estimate f over whole $\mathcal{X}$. In this setting the problem is called *density estimation* problem (see [Vap98, Bis95]).

A more general setting of the density estimation problem is when x-es are drawn according to multiple distributions $f_1, .., f_k$, when the generating distribution is chosen randomly before each trial or sequence of trials. If in addition to learning the resulting *mixture distribution* we try to learn each f_j in particular, the problem is known as a problem of *unsupervised learning* (unsupervised since we do not get the correspondence between the data points and their generating distributions explicitly in our input). When our primary interest focuses on finding the correspondences between the data points and the sources $(f_1, .., f_k)$ that most likely generated them, the problem is also known as *unsupervised classification* or *clustering* of the data (here the class of a point is the index of the distribution function it was most likely sampled from).

The problem of unsupervised learning was deeply studied for the case of independent random variables in $\Re^n$ (points) - see [DHS01] and [Ros98] for an overview. Though little work was done for the case of dependent variables and sequences in particular (see [FR95]).

At the same time 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 (see [Rab86]). HMMs are predefined parametric models - their architecture and topology are predetermined and the memory is limited to first order in most common applications. The success of HMMs thus crucially depends on the correct choice of the state model. It is rather difficult to generalize these models to hierarchical structures with unknown a-priory state topology (see [FST98] for an attempt).

An interesting alternative to the HMM was proposed in [RST96] in the form of a sub class of *probabilistic finite automata*, the variable memory Markov (VMM) sources. These models have several important advantages over the HMMs:

1. They capture longer correlations and higher order statistics of the sequence.
2. They can be learned in a provably optimal PAC like sense using a construction called *prediction suffix tree (PST)* [RST96].
3. They can be learned efficiently by linear time algorithm [AB00].
4. Their topology and complexity are determined by the data.

In this work we describe a powerful extension of the VMM model and the PST algorithm to a stochastic mixture of such models, suggested in [SBT01b] and present a detailed analysis of the algorithm. The problem we are trying to solve is: given a string $\bar{x} = x_1..x_n$ that was generated by repeatedly switching between a number of unknown VMM sources (with some upper bound on the alternation rate), find the most likely number of sources that participated in the generation of $\bar{x}$ and the most probable segmentation of $\bar{x}$ into segments, generated by each of the sources. The problem is generally computationally hard, similarly to data clustering. Only very simple sequences can be segmented both correctly and efficiently in general (see [FR95, Hof97]).

We approach this problem with hierarchical top-down clustering procedure. Our approach is information theoretic in nature. The goal is to enable short description of the data by a (soft) mixture of VMM models, each one controlled by an MDL principle (see [BRY98] for a review). The last is done by modifying the original PST algorithm using the MDL formulation, while

preserving its good learnability properties. The mixture model is then learned via a generalized *rate distortion theory* approach (see [CT91, Ch. 13]). 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 [CT91]) 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 (see [Ros98]) 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. 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.

After describing and analyzing the algorithm we demonstrate an interesting application of the algorithm in the field of protein sequences classification. This application was widely explored in [BSMT01], which was a natural continuation of the [BY01] work, where PSTs were shown to be a powerful tool for supervised classification of proteins. The current work extends our abilities by allowing to perform this task in unsupervised manner. Characterization of a protein sequence by its distinct domains (autonomic structural subunits) is crucial for correct classification and functional annotation of newly discovered proteins. Many families of proteins that share a common domain contain instances of several other domains without any common ordering, nor with mandatory presence of the additional domains. Therefore, conventional multiple sequence alignment (MSA) methods (that attempt to align the complete sequence, see [DEKM98]) find difficulties when faced with heterogeneous groups of proteins. Their success crucially depends on the initial (seed) selection of a group of related proteins, usually hand crafted by experts. Even in the cases when similarities are detected in an automatic way using bottom-up clustering techniques [Yon99], the system lacks the global picture view. The advantage of our algorithm is that it does not attempt any alignment, but rather clusters together short regions with similar statistics. As a result it does not require any initial selection of a group of related proteins, and it is not confused by different orderings of the domains in the protein sequences. The classification is done through revelation of domain signatures - short, highly conserved domain subsegments common to at least small amount of the input proteins.

The continuation of the work is built in the following way. In Ch. 2 we give some basics from the probability, information and learning theories, essential for understanding of our work. In Ch. 3 we define variable memory Markov (VMM) processes. We then review the algorithms of [WST95] and [RST96] for learning of VMM sources and describe the new algorithm from [SBT01b] that approaches this task basing on the MDL principle. Ch. 4 gives an introduction to the field of unsupervised learning of mixture distributions and reviews some clustering algorithms. In Ch. 5 we describe the new algorithm for unsupervised sequence segmentation from [SBT01b]. The algorithm embeds the VMM sequence modelling described in Ch. 3 into hierarchical clustering framework described in Ch. 4. We also hold a short discussion of the main points of the algorithm at the end of Ch. 5. Ch. 6 demonstrates two interesting applications of the algorithm. The first one is unsupervised segmentation of multilingual texts into languages. Here the algorithm was able to infer correctly both the number of languages used and the language switching points with a precision of a few letters. We also try to sense the limitations of

the algorithm on this example in the notion of maximal switching rate it is able to detect and minimal amount of data it needs. The second application shown is unsupervised classification of protein sequences. Here the algorithm was able to refine the HMM superfamily classification and to identify domains that appeared in a very small amount of the input proteins. The section includes the results of [SBT01b, BSMT01, SBT01a] works, as well as some new results first presented here (mainly the analysis of the abilities of the algorithm) and some results that did not enter our previous papers due to space limitations. Ch. 7 holds a discussion of the algorithm and the results and gives a number of suggestions for further work.

Chapter 2

Preliminaries

2.1 Essential Concepts from the Probability Theory

In this section we are going to give a number of essential definitions from the probability theory.

Conditional Probability and Bayes Formula

Conditional probability is one of the most basic instruments in the probability theory and will be extensively used in this work. We start with an illustrative example and then will give a formal definition.

Suppose that we have a population of N people - N_M men and N_W women. And suppose that N_R out of them have read this work. We denote by M, W and R the events that a person is a man, a woman and has read this work respectively. Then (see [Fel71]) $P(M) = \frac{N_M}{N}$, $P(W) = \frac{N_W}{N}$ and $P(R) = \frac{N_R}{R}$. Now we can concentrate on the subset of our population containing women only and ask, what is the probability that a randomly chosen women has read this work. We denote the probability of this event by $P(R|W)$, which can be read as: "the probability of event R conditioned on event W" or "the probability of (event) R given (event) W". If N_{WR} is the number of women who has read this work, then:

$$P(R|W) = \frac{N_{WR}}{N_W}$$

on the other hand:

$$\frac{N_{WR}}{N_W} = \frac{\frac{N_{WR}}{N}}{\frac{N_W}{N}} = \frac{P(W \cap R)}{P(W)}$$

which means:

$$P(R|W) = \frac{P(W \cap R)}{P(W)}$$

(Later we will also use a notation $P(W, R)$ for the probability of the intersection of the events W and R.)

This brings us to the following definition:

Definition 2.1 *Let H be an event with positive probability. Then for every event A we write:*

$$P(A|H) = \frac{P(A \cap H)}{P(H)} \tag{2.1}$$

Note, that working with conditional probabilities for a given fixed event H is equivalent to choosing H as our new space of elementary events with probabilities proportional to the original ones - $P(H)$ plays here the role of normalization coefficient. This means that all the basic theorems on probabilities are valid for conditional probabilities as well. For example: $P(A \cup B|H) = P(A|H) + P(B|H) - P(A \cap B|H)$.

(2.1) may be rewritten in the form:

$$P(A \cap B) = P(A|B) \cdot P(B) \tag{2.2}$$

This may be generalized for a sequence of events $A_1, .., A_n$:

$$\begin{aligned} P(A_1 \cap .. \cap A_n) &= P((A_1 \cap .. \cap A_{n-1}) \cap A_n) \\ &= P(A_1 \cap .. \cap A_{n-1}|A_n) \cdot P(A_n) = ... \\ &= P(A_1|A_2 \cap .. \cap A_n) \cdot P(A_2|A_3 \cap .. \cap A_n) \cdot .. \cdot P(A_{n-1}|A_n) \cdot P(A_n) \end{aligned} \tag{2.3}$$

Such chain decomposition of the probability will be very useful when we get to the Markov processes.

Now we give one more definition that will be used in this chapter:

Definition 2.2 *Two random variables A and B will be called* independent, *if $P(A|B) = P(A)$.*

Note, that if $P(A|B) = P(A)$, then $P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{P(B) \cdot P(A|B)}{P(A)} = \frac{P(B) \cdot P(A)}{P(A)} = P(B)$.

Now let us take a set of non-intersecting events $H_1, .., H_n$, such that $\bigcup_{i=1}^{n} H_i$ covers the whole probability space. This means that every event belongs to a single H_i out of $H_1, .., H_n$. In this case, for any event A, $A = \bigcup_{i=1}^{n} (A \cap H_i)$. Using the fact that for $i \neq j$ we have $P((A \cap H_i) \cap (A \cap H_j)) = 0$ due to the emptiness of the intersection of H_i with H_j we get:

$$P(A) = P(\bigcup (A \cap H_i)) = \sum P(A \cap H_i) = \sum P(A|H_i) \cdot P(H_i) \tag{2.4}$$

From here we straightly get the *Bayes formula*:

$$P(H_j|A) = \frac{P(A \cap H_j)}{P(A)} = \frac{P(A|H_j) \cdot P(H_j)}{\sum_i P(A|H_i) \cdot P(H_i)} \tag{2.5}$$

If $\{H_i\}$ is our *hypothesis set*, then $P(H_i)$ is called the *prior* probability distribution over the hypothesizes and $P(H_i|A)$ is called the *posterior* distribution over the hypothesizes - after we know that A has happened.

Note, that if H_i-s are the states of our world, and A is some observation we have done, then we can infer some information about the state of the world we are currently in. For example, we have two unfair coins: C_1 has a greater probability for "head" and C_2 has a greater probability for "tail". Suppose that we have chosen one out of the two coins according to some prior distribution $P(C_i)$ and made a trial. Then according to the result we got, we can tell what is the posterior probability that we have chosen C_i.

Probability Density

When talking about continuous variables one should consider probability density functions. A probability density function $p(x)$ specifies that the probability of the random variable $X \in \mathcal{X}$ lying in the region $R \subseteq \mathcal{X}$ is given by:

$$P(X \in R) = \int_R p(x)dx$$

(2.1) may be generalized for the density functions. Let $p(x, y)$ be the joint probability density function of two random variables $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$:
$P((X \in R_X) \wedge (Y \in R_Y)) = \int_{R_X, R_Y} p(x, y)dxdy$. And let $p(y)$ be the *marginal* density function of Y: $p(y) = \int_{\mathcal{X}} p(x, y)dx$. Then denoting by $f_y(X)$ the probability density of the random variable $\{X|Y = y\}$, we get:

$$f_y(X) = \frac{p(X, y)}{p(y)}$$

See [Fel71] for a proof.

Conditional Expectation

Now we add a notion of conditional expectation.

Definition 2.3 Conditional Expectation $E(Y|X = x)$ *is defined as:*

$$E(Y|X = x) = \sum_{y \in \mathcal{Y}} yp(y|x)$$

We write $E(Y|X)$ when we talk about the conditional expectation as a function of X, and $E(Y|x)$ when we talk about its value at specific point x. Note, that when we talk about conditional expectation we assume an existence of the joint probability distribution $p(x, y)$.

Jensen's Inequality

Theorem 2.1 *(Jensen's inequality): If f is a convex function and X is a random variable, then:*

$$Ef(X) \geq f(EX)$$

Moreover, if f is strictly convex, then equality implies that $X = EX$ with probability 1, i.e. X is a constant.

See [CT91, page 25] for a proof.

2.2 Information Theory

Information Theory originated from the Communication Theory in the early 1940's and initially dealt with the questions of data compression and transmission. The first and most important results are due to Shanon, who actually founded this field of studies ([Sha48] and later works). Though being still associated with Communication Theory, Information Theory proved to have

important relations to other fields of study, such as Thermodynamics in Physics, Kolmogorov Complexity in Computer Science, Economics, Probability Theory, Statistics and Machine Learning. Here we will focus on the last one, while all the rest, as well as a good reference to the whole theory may be found in [CT91].

One of the most basic quantities in the information theory is the *entropy* of a distribution:

Definition 2.4 *The* entropy $H(X)$ *of a discrete random variable* X *distributed according to* p *is defined by:*

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) = E - \log p(x) \tag{2.6}$$

We will also write $H(p)$ for the above quantity. In this work we will only use binary entropy, i.e. the log in the definition of H is $\log_2$, also denoted as lg.

Another important quantity we want to define here is *relative* or *cross entropy*, also known as *Kullback-Leibler distance* or *divergence*, suggested in [KL51]:

Definition 2.5 *The* Kullback-Leibler distance *between two probability distributions* $p(x)$ *and* $q(x)$ *is defined as:*

$$D_{KL}(p\|q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} = E_p \log \frac{p(x)}{q(x)} \tag{2.7}$$

In the above definition we use the convention (based on continuity arguments) that $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$. Note, that D_{KL} is asymmetric ($D_{KL}(p\|q) \neq D_{KL}(q\|p)$) and does not satisfy the triangle inequality. Actually, the only property of a metric it satisfies is positivity (see [CT91, page 26] for a proof):

Theorem 2.2 *(Information inequality): For probability distributions* p *and* q, $D_{KL}(p\|q) \geq 0$ *with equality if and only if* $p(x) = q(x)$ *for all* x.

Nonetheless, it is often useful to think of relative entropy as a "distance" between distributions for reasons that will be immediately shown after a sequence of definitions related to coding:

Definition 2.6 *A* source code C *for a random variable* X *is a mapping*
$C : \mathcal{X} \to \{0,1\}^*$. $C(x)$ *denotes the codeword corresponding to* x *and* $l_C(x)$ *denotes the length of* $C(x)$.

The subscript C in $l_C(x)$ will be omitted wherever it will be clear which code C do we speak about. In this work we will dial solely with binary codes.

Definition 2.7 *The* expected code length $L(C)$ *of a source code* C *is given by:*

$$L(C) = \sum_{x \in \mathcal{X}} p(x) l(x) \tag{2.8}$$

Definition 2.8 *Code* C *is called* uniquely decodable *if for every two sequences* $x_1..x_n, y_1..y_m \in \mathcal{X}^*$ *such that* $x_1..x_n \neq y_1..y_m$:
$C(x_1..x_n) = C(x_1)..C(x_n) \neq C(y_1)..C(y_m) = C(y_1..y_m)$.

With above definitions it may be shown that (see [CT91] for a proof):

Theorem 2.3 *(McMillan): The codeword lengths of any uniquely decodable code must satisfy the Kraft inequality:*

$$\sum_{x\in\mathcal{X}} 2^{-l(x)} \leq 1$$

If we try to minimize $L(C)$ while constrained by the Kraft inequality, we derive (using the technique of Lagrange multipliers) that the optimal code lengths l^* should satisfy: $l^*(x) = -\lg p(x)$. Which implies: $L(C) \geq L(C^*) = -\sum p(x)\lg p(x) = H(X)$, where C^* stays for the optimal code. We deduced that $H(X)$ is the lower bound on the code length of X. This is also an achievable bound, and there are algorithms (like Huffman code) that achieve: $H(X) \leq L(C) < H(X)+1$. All these codes satisfy: $l(x) \leq \lceil -\lg p(x) \rceil$.

Note, that if we construct a code C for X using a "wrong" distribution $q \neq p$, we get:

$$L(C) = E_p l_q(x) = -\sum_{\mathcal{X}} p(x)\lg q(x)$$

$$= -\sum_{\mathcal{X}} p(x)\lg p(x) + \sum_{\mathcal{X}} p(x)\lg\frac{p(x)}{q(x)} = H(p) + D_{KL}(p\|q)$$

Thus $D_{KL}(p\|q)$ is the penalty *per symbol* we will pay for choosing a wrong distribution q when trying to code a sequence generated according to p. This gives us the motivation for taking D_{KL} as a measure of distance between distributions.

Another information theoretic quantity we want to define here is *mutual information*:

Definition 2.9 *For two random variables X and Y with joint probability distribution $p(x,y)$ and marginal distributions $p(x)$ and $p(y)$ the* mutual information $I(X;Y)$ *is the relative entropy between the joint and the product distribution $p(x)p(y)$:*

$$I(X;Y) = \sum_{x\in\mathcal{X}, y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

$$= D_{KL}(p(x,y)\|p(x)p(y))$$

We add a definition of *conditional entropy*:

Definition 2.10 *For two discrete random variables X and Y with a joint distribution $p(x,y)$ the* conditional entropy $H(Y|X)$ *is defined as:*

$$H(Y|X) = \sum_{x\in\mathcal{X}} p(x)H(Y|x)$$

$$= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x)$$

With this definition we note that:

$$I(X,Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

$$= \sum_{x,y} p(x)p(y|x) \log \frac{p(y|x)}{p(x)}$$

$$= -\sum_{x} p(x) \log p(x) + \sum_{x} p(x) \sum_{y} p(y|x) \log p(y|x)$$

$$= H(X) - H(X|Y)$$

Thus the mutual information $I(X;Y)$ is the reduction in the uncertainty of X due to the knowledge of Y.

2.3 Probability Density Estimation

One of the main problems the learning theory deals with and that will be touched in this work is the problem of probability density estimation. The usual setting for this problem is: given a finite sample $\mathbf{x} = x_1, .., x_n$ of independent samples generated by unknown probability distribution (source) $p(x)$, try to estimate $p(x)$ for the whole probability space $\mathcal{X}$.

There are three major approaches to the density estimation problem. The *parametric* methods, in which a specific functional form for the density model is assumed, i.e. $p(x) = f_\theta(x)$, where f is some function and θ is its parameters vector belonging to a parameters space Θ. For example, for a normal distribution $\theta = (\mu, \sigma) \in \Re \times \Re^+ = \Theta$. The drawback of this approach is that the particular form of parametric function chosen might be incapable of providing a good representation of the true density. A different approach is *non-parametric* estimation which does not assume a particular form, but allows the form of the density to be determined entirely by the data. Such methods typically suffer from the problem that the number of parameters in the model grows with the data set, so that the model can quickly become unwieldy. The third approach, sometimes called *semi-parametric* estimation, tries to achieve the best of both worlds by allowing a very general class of functional forms in which the number of parameters can be increased in a systematic way to build even more flexible models, but where the total number of parameters can be varied independently from the size of the data set. For example, the PST model in Sec. 3.2.1 is a semi-parametric one.

2.3.1 Maximum Likelihood

Being the most straightforward approach to the density estimation, the parametric method assumes that the unknown probability density may be represented in terms of specific functional form which contains a number of adjustable parameters. Namely, we think about some function $f(x, \theta)$ such that for each *fixed* θ, $f_\theta(x)$ represents a probability distribution, and then we say that $p(x|\theta) = f_\theta(x)$, where θ is the set of (unknown) parameters. There are two principal techniques for determining the unknown parameters θ of the distribution given a sample $\mathbf{x}$ of independent samples generated according to $p(x|\theta)$. The first one, *maximum likelihood* is discussed here.

To be able to talk about maximum likelihood we define the *likelihood* of a sample $\mathbf{x}$ under fixed parameters set θ (and a fixed model f we are investigating):

$$\mathcal{L}(\mathbf{x}) = \mathcal{L}(x_1, .., x_n|\theta) = P(x_1, .., x_n|\theta) = \prod_{i=1}^{n} p(x_i|\theta)$$

The last equality holds because $x_1, .., x_n$ are generated independently by $f_\theta(x)$ and are therefore independent given θ.

The maximum likelihood technique just chooses $\theta_{ML} = argmax_\theta \mathcal{L}(\mathbf{x})$ to be the estimation of the unknown distribution parameter θ. In words, the most probable parameter θ that was involved in generation of the observed sample is the one that maximizes the likelihood of the sample.

2.3.2 Bayesian Inference

Now we turn to describe the second technique for parameter estimation, named *Bayesian Inference*. We note, that if we choose $p(\theta)$ to be our *prior* distribution over the parameters space Θ, then $p(\mathbf{x}, \theta) = p(\theta)p(\mathbf{x}|\theta)$ is a legal probability distribution over the $\mathcal{X}^n \times \Theta$ space. $p(\theta)$ represents our uncertainty in the values of the unknown parameters θ. Before we see a new sample $\mathbf{x}$, its prior probability in our model is (due to continuous version of (2.4)):

$$p(\mathbf{x}) = \int_\Theta p(\mathbf{x}|\theta) \cdot p(\theta) d\theta$$

After we saw the sample our *posterior* distribution over the parameters $p(\theta|\mathbf{x})$ becomes (by continuous version of (2.5)):

$$P(\theta|\mathbf{x}) = \frac{p(\theta) \cdot p(\mathbf{x}|\theta)}{p(\mathbf{x})}$$

Unlike maximum likelihood, which gives us a specific value of θ, Bayesian inference provides us with a posterior distribution over the parameters space Θ. This distribution may be then used to predict new samples. By (2.4) we have:

$$p(X|\mathbf{x}) = \int_\Theta p(X|\theta, \mathbf{x}) \cdot p(\theta|\mathbf{x}) d\theta = \int_\Theta p(X|\theta) \cdot p(\theta|\mathbf{x}) d\theta$$

Where the second equality holds due to independence of X and $\mathbf{x}$ given θ.

2.3.3 Probably Approximately Correct Learning Model

Being probably the most natural learning approach, Bayesian inference does not provide us (at least directly) with any guaranties on the quality of the answer we found. I.e. we know that we found the most likely approximation of the unknown parameter θ, but we have no idea of how far we are from the actual value of θ that generated the sample. Probably Approximately Correct (PAC) model takes this question as a starting point.

PAC learning model was suggested by Valiant in [Val84]. The idea of PAC is to find the hypothesis that with high probability will not be too far from the target one. In our context of probability density estimation we will be mainly concerned with the Kullback-Leibler distance between distributions. We will say that:

Definition 2.11 *A family $\mathcal{P}$ of probability distributions over $\mathcal{X}$ is* PAC-learnable*, if there exists an algorithm $\mathcal{A}$ that for every unknown distribution $p \in \mathcal{P}$, given a sufficient, but at most polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$, amount of samples generated according to p, provides a hypothesis q that satisfies: $P(D_{KL}(p\|q) > \epsilon) < \delta$.*

We leave $\mathcal{A}$ the possibility to err completely with probability at most δ since the sample generated by p may appear to be untypical to p. For example, a fair coin may, though with small probability, occasionally generate a long sequence of "all ones".

Main results on the possibility of PAC learning are based on Glivenko-Cantelly theorem on the convergence of empirical distributions to the actual distribution function [Gli33, Can33]. The sample efficiency of the learning procedure (in appropriate cases) is based on Kolmogorov's work on the rate of that convergence [Kol33] and Chernoff and Hoeffding inequalities [Che52, Hoe63]. The necessary and sufficient conditions for PAC learning are based on the work of Vapnik and Chervonenkis [VC71, VC81]. A much more detailed discussion of PAC may be found in [KV94] or [Vap98].

2.3.4 Minimum Description Length Principle

In Sec. 2.2 we saw that the optimal code length mimics data generating distribution; namely $l(x) = -\lg p(x)$. Though, if we were wishing to transmit a sequence $x_1, .., x_n$ we would have to specify which code C we are using as well. Thus the total number of bits we transmit would be:

$$l(C) + \sum_{i=1}^{n} l_C(x_i) \tag{2.9}$$

where $l(C)$ stays for the length of specification of C.

Note, that if we choose C out of a large family of codes $\mathcal{C}$, then we can find C built on the underlying distribution q_C which will be very close to the data generating distribution p. In such case, we will pay low penalty $nD_{KL}(p\|q)$ for not being absolutely exact in our estimation of p, but the specification of C will be rather long (if there is an equal probability for choosing any $C \in \mathcal{C}$, the specification of C will take $\lg|\mathcal{C}|$ bits). This would be especially inefficient if we have small amount of data n. For hierarchical families of distributions (when subsequent probability density functions refine the partition of their "parents") it is possible that different hypotheses q_C will have different specification lengths $l(C)$; see Sec. 3.2.2 for such example.

Minimum Description Length principle stays that the optimal hypothesis q_{C^*} is the one that underlies the code C^* that minimizes (2.9).

It may be further shown that:

1. If we use Bayesian inference, then $P(x_{n+1}|x_1, .., x_n) \approx P(x_{n+1}|q_{C^*})$ up to $o(1)$, thus MDL is a good approximation of the Bayesian inference (see [BRY98]).

2. In PAC learning setting q_{C^*} minimizes the risk that the prediction will (significantly) disagree with the actual process outcome (see [Vap98] for a proof for pattern recognition problem).

MDL approach has several advantages over the inference schemes described previously. If compared to Bayesian inference, MDL is a good approximation of Bayesian inference, but at the

same time MDL final hypotheses are usually much more compact than Bayesian ones. In addition, MDL provides us with a single model and not a mixture of models, which sometimes is also an advantage. If compared to PAC learning setting, MDL suggests only a single parameter for optimization - the description length, that comes instead of ϵ and δ parameters of PAC learning algorithms, and still it reaches the same goal - minimization of probability of error. Therefore MDL is more suitable for unsupervised learning frameworks, where we want to minimize the number of parameters externally controlled by the user. MDL is also an ultimative tool for revealing the actual data generating distribution p since q_{C^*} converges to p as the sample size n tends to infinity.

MDL principle was suggested in the work of Rissanen [Ris78], though it should be noted that very similar ideas appeared also in preceding works, like [WB68]. MDL principle has very tough relations to the Kolmogorov complexity, defined in the works of Solomonoff [Sol60], Kolmogorov [Kol65] and Chaitin [Cha66]. A good reference to MDL principle is [BRY98]. [Vap98] suggests some additional bounds from the risk minimization point of view.

Chapter 3

Markov Processes and Sequential Data Sources

This chapter is devoted to single sequential source modeling. We start with definition of Markov and variable memory Markov (VMM) processes. Then we review some existing algorithms for learning of VMM sources and finish with detailed description of the new VMM source learning algorithm from [SBT01b].

3.1 Markov and Variable Memory Markov Processes

For a sequence of random variables $\bar{X} = X_1..X_n$ we may use (2.3) to write the probability of the sequence in the following way:

$$P(\bar{X}) = \prod_{i=1}^{n} P(X_i|X_1..X_{i-1})$$

In Ch. 2 we assumed that $\{X_i\}_{i=1}^n$ are independent variables, i.e. $P(X_i|X_1,..,X_{i-1}) = P(X_i)$. But in many cases this assumption appears to be too strict.

The Markov assumption is less restrictive (see [Fel71] for a deeper discussion):

Definition 3.1 *A sequence of random variables is said to form a* Markov chain, *if* $P(X_i|X_1,..,X_{i-1}) = P(X_i|X_{i-1})$.

Here X_{i-1} represents the chain "memory", which in the particular case of the definition is of length 1. Alternatively, we may assume that $P(X_i|X_1,..,X_{i-1}) = P(X_i|X_{i-r},..,X_{i-1})$, getting memory of length r. The problem is that for a sequence over an alphabet of size $|\Sigma|$ and memory of length r, the number of conditional distributions we will have to learn is $|\Sigma|^r$, limiting us due to sample size or space constrains to very short memory length, which is not always sufficient to capture all significant long-distance dependencies in the data. The key for success of the VMMs lies in the observation that out of $|\Sigma|^r$ possible "memories" usually only few are likely to frequently appear. For the rest we can suffer even a big mistake (loss) since the number of times this will happen will be small.

VMM source may be represented using a tree of selected suffixes of the prefixes $\{x_1..x_{i-1}\}_{i=1}^n$. In [RST96] such tree is called Prediction Suffix Tree (PST). In [WST95] the string $x_{i-r}..x_{i-1}$ is called a *context* of x_i, and the tree is called Context-Tree. We will use the following definition:

Definition 3.2 *A* Prefix-Suffix Tree T *over a finite alphabet* Σ *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*.*

See Fig. 3.1 for an illustration of such tree.

Definition 3.3 $suf_T(x_1..x_{i-1})$ *is defined to be the longest sequence* $x_{i-r}..x_{i-1}$ *that makes a path in* T *in the following sense: we start from the root and traverse the edge labeled by* x_{i-1}*, from there we traverse the edge labeled by* x_{i-2} *etc., until there is no appropriate edge to continue with or we have traversed the whole string*[1]*. If there is no edge labeled by* x_{i-1} *leaving the root we say that* $suf_T(x_1..x_{i-1}) = \lambda$*.*

Definition 3.4 *A* Variable Memory Markov (VMM) *source* G *is a stochastic process that satisfies:* $P_G(x_i|x_1..x_{i-1}) = P_G(x_i|suf_T(x_1..x_{i-1}))$ *for some suffix tree* T*.* T *will be called a* supporting tree *of* G*.*

Definition 3.5 *The* Minimal Supporting Tree (MST) *of a VMM source* G *is a supporting tree of* G *that satisfies: for all* $T' \subset T$*,* T' *is not a supporting tree of* G[2]*.*

Theorem 3.1 *For each VMM source exists a* unique *MST.*

Proof 3.1 *Let* G *be a VMM source. Then, by definition,* G *has a supporting tree* T*. There is a finite number of trees* $T' \subset T$*, thus a minimal one exists. It is left to show the uniqueness of the minimal tree. Suppose that* T_1 *and* T_2 *are two different MSTs of* G*. Then:*

1. $T_1 \cap T_2$ *is a supporting tree of* G*, since:*

$$\begin{cases} P_G(x_i|x_1..x_{i-1}) = P_G(x_i|suf_{T_1}(x_1..x_{i-1})) \\ P_G(x_i|x_1..x_{i-1}) = P_G(x_i|suf_{T_2}(x_1..x_{i-1})) \end{cases}$$

$$\begin{aligned} \Rightarrow P_G(x_i|x_1..x_{i-1}) &= P_G(x_i|min\{suf_{T_1}(x_1..x_{i-1}), suf_{T_2}(x_1..x_{i-1})\}) \\ &= P_G(x_i|suf_{T_1 \cap T_2}(x_1..x_{i-1})) \end{aligned}$$

2. $T_1 \cap T_2 \subset T_1$ *and* $T_1 \cap T_2 \subset T_2$*, contradicting the minimality of both.*

□

In this work we assume that all sources are stationary and ergodic since these two requirements are essential for any learning be possible.

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

[2] $T \supset T'$ if T may be obtained from T' by addition of nodes.

Definition 3.6 *A VMM process is called* stationary, *if for each* $i, j \geq 1$,
$P(X_i = \sigma | suf_T(x_1..x_{i-1})) = P(X_j = \sigma | suf_T(x_1..x_{j-1}))$
whenever $suf_T(x_1..x_{i-1}) = suf_T(x_1..x_{j-1})$.

Definition 3.7 *A VMM process is called* ergodic, *if for each pair of strings* s, t, *such that* $P(X_1..X_{|s|} = s) > 0$, $P(X_l..X_{l+|s|} | x_1..x_{|t|} = t) > 0$ *for some finite* $l > |t|$.

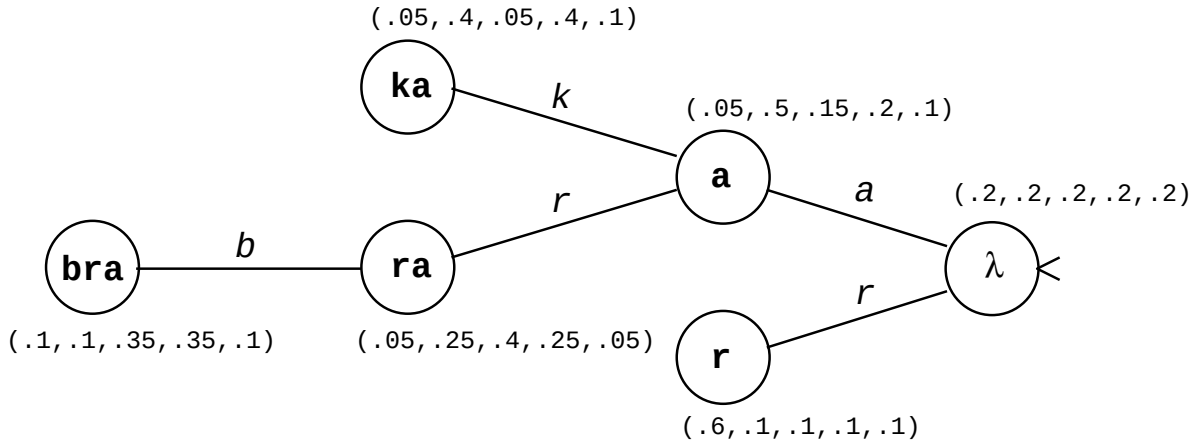


Figure 3.1: **An example of a PST** over the alphabet $\Sigma = \{a, b, k, l, r\}$. The vector near each node is the probability distribution for the next symbol. E.g., the probability to observe k after the substring *bara*, whose largest suffix in the tree is *ra*, is $P(k|bara) = P_{ra}(k) = 0.4$.

A natural way to model VMM source is a PST:

Definition 3.8 Prediction Suffix Tree (PST) T *is a Prefix-Suffix Tree with the following property:*

1. *A probability distribution vector over* Σ *is associated with each node* $s \in T$.
   $P_s(\sigma) \equiv P(\sigma|s)$ *is the probability that letter* σ *will come after a string* s.

See Fig. 3.1 for an illustration of PST.

It may be shown that when $P_s(\sigma)$ is defined to be $P_s(\sigma) = \frac{\text{the number of times } \sigma s \text{ occurred in } \bar{x}}{\text{the number of times } s \text{ occurred in } \bar{x}}$ for some string $\bar{x}$ and with proper handling of the end points of $\bar{x}$, the distributions $P_s(\sigma)$ satisfy the marginal condition: $P_s(\sigma) = \sum_{\hat{\sigma} \in \Sigma} \frac{P_T(\hat{\sigma}s)}{P_T(s)} P(\sigma|\hat{\sigma}s)$, where $P_T(s)$ is as defined below and $\hat{\sigma}s$ is a prefix extension of s (see [RST96]). This means that the complete set of distributions $P_s(\sigma)$ is stationary.

Predicting and Generating using PSTs

Here we define the probability measure that a PST T induces on the space of all strings $\bar{x} = x_1..x_n \in \Sigma^n$, 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 $P_{suf_T(x_1..x_{i-1})}$.

3.2 Learning Variable Memory Markov Sources

In this section we review the algorithms from [RST96] and [WST95] for VMM source learning and then describe the new algorithm from [SBT01b] that will be used in this work.

3.2.1 PAC Learning of the VMM Sources

In [RST96] a PAC algorithm for VMM source learning was proposed. The algorithm works in the following way. As an input it gets a string $\bar{x}$ (or a set of strings) generated by the explored source, precision and confidence parameters ϵ, δ and maximal assumed depth of the MST of the generating source L. For each substring s of length $|s| \leq L$ the empirical probability of s, $\tilde{P}(s)$, is defined to be the number of times s appeared in $\bar{x}$ divided by $|\bar{x}| - L - 1$ - the number of times s could appear in $\bar{x}$. For each letter $\sigma \in \Sigma$ the empirical probability of σ to come after s, $\tilde{P}_s(\sigma)$, is defined to be the number of times σ appeared after s in $\bar{x}$ divided by the number of occurrences of s. The output of the algorithm is a PST that with probability of at least $1 - \delta$ is ϵ-close (in the D_{KL} pseudo-metrics) to the original source. The final tree is a collection of all nodes that satisfy:

1. $|s| \leq L$
2. $\tilde{P}(s)$ is greater than some lower bound that is a function of ϵ, δ and $|\Sigma|$.
3. For each node $\hat{\sigma}s$ there is $\sigma \in \Sigma$ such that $\tilde{P}(\sigma|\hat{\sigma}s)$ differs significantly from $\tilde{P}(\sigma|s)$, or there is a descendant $\hat{s}\hat{\sigma}s$ of $\hat{\sigma}s$ for which $\tilde{P}(\sigma|\hat{s}\hat{\sigma}s)$ differs significantly from $\tilde{P}(\sigma|s)$. The significance is a function of ϵ, δ and $|\Sigma|$.

Smoothing of probability distributions in the nodes of the final tree T is done to avoid zero probabilities. The PAC property of the algorithm is proved in [RST96]. A linear time and space algorithm to find T was proposed in [AB00].

3.2.2 Bayesian Learning of the VMM Sources

In [WST95] a Bayesian approach to VMM source learning was proposed. The input to the algorithm is a binary string generated by the explored source (generalization to finite alphabet is discussed elsewhere) and the assumed maximal depth L of the MST of that source (this assumption was eliminated in [Wil98]). The output is a weighted combination of all possible context trees of depth not greater than L (over all possible trees in [Wil98]).

The prior probability of a tree T is inverse proportional to the exponent of the description length of T. In their coding scheme, [WST95] for each node s of T code the existence of sons of s: for each $\hat{\sigma} \in \Sigma$ the bit of $\hat{\sigma}$ in s is 1, if $\hat{\sigma}s \in T$ and 0 otherwise[3]. Thus the description length of the tree skeleton is $|\Sigma| \cdot |T|$, where $|T|$ is the number of nodes in T. For each tree T and for each node s, $\tilde{P}_s(\sigma)$ is defined to be: $\tilde{P}_s(\sigma) = \frac{\tilde{N}_s(\sigma)+\frac{1}{2}}{\tilde{N}(s)+\frac{1}{2}|\Sigma|}$, where $\tilde{N}(s)$ is the empirical number of

[3] $s \in T$ means that s is a node in T

occurrences of s in $\bar{x}$, $\tilde{N}_s(\sigma)$ is the empirical number of times σ appeared after s in $\bar{x}$ and $\frac{1}{2}$ comes from the usage of Krichevsky-Trofimov (KT) estimators (see [KT81]) which ensure good bounds on the distance between $\tilde{P}_s(\sigma)$ and the real distribution $P_s(\sigma)$ for small sample sizes.

All the trees are stored in one complete $|\Sigma|$-ary tree of depth L. An efficient procedure for simultaneous update of $\tilde{P}_s(\sigma)$ for all the trees as well as an efficient procedure for weighting of the predictions of all the trees is described in [WST95]. Both procedures run in time linear in L.

It is shown in [Wil98] that when no assumptions on the tree depth are made (i.e. we use the algorithm from [Wil98]) or when L is greater or equal to the depth of the MST of the generating process, the entropy $H(P_T(X_i|X_1..X_{i-1}))$ converges to the entropy of the generating source with probability 1 as the sample size n tends to infinity.

3.2.3 MDL Learning of the VMM Sources

Now we turn to describe the new MDL driven algorithm for PST training from [SBT01b]. The algorithm has the advantages of both PAC and Bayesian algorithms we have just described:

1. The algorithm gets no parameters and thus perfectly suits for unsupervised learning that will be discussed starting from the next chapter.

2. The resulting (and intermediate) tree is very compact. Thus it is very handful for work with strings over large alphabets and in the cases when we have multiple models (see Ch. 5).

   In addition:

3. Being built on MDL principles, the algorithm reveals the most likely MST of the generating process, which may be interesting on its own.

4. The algorithm was generalized to handle weighted data. This extension will be necessary when we will start working with multiple models, but it may be also useful for single model setting in the cases when we have different levels of confidence in our input data.

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 $(0 \leq w_i \leq 1)$[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 $w_i = 1$ (this corresponds to the simple counting setting we had in the previous two algorithms).

For a string s we say that $sx_i \in \bar{x}$ if sx_i 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

$$w(s) \equiv \sum_{\sigma \in \Sigma} w_s(\sigma)$$

[4]Generalization to a set of strings is straightforward and therefore omitted here for ease of notation. See [RST96] for an example of such generalization on the original algorithm.

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 3.2: The PST learning algorithm.

Clearly $\frac{w_s(\sigma)}{w(s)}$ is an empirical estimate for $P_s(\sigma)$. We smooth the probabilities by using the KT-estimators for the same reasons as in [WST95].

As described in Ch. 2, the idea behind the 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. The coding of the tree skeleton takes $|\Sigma|$ bits per node as in [WST95]. In addition we should code the probability distribution vectors $\{P_s(\sigma)\}_{s \in T}$. Note, that the distribution vector P_s is used to code only those x_i-s, for which $suf_T(x_1..x_i) = s$. Thus the total amount of data that is coded using P_s is at most $w(s)$, and exactly $w(s)$ for the leaf nodes. In order to achieve minimal description length of the vector P_s together with the fraction of the data that is coded using P_s the counts $w_s(\sigma)$ should be coded to within accuracy of $\sqrt{w(s)}$ (see [BRY98]). Each node s holds $|\Sigma|$ of such counts, thus the description size of s is:

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

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

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

When coding data "passing through" node s [5]:

1. If $suf_T(x_1..x_{i-1})$ ends with $\hat{\sigma}s$ for $\hat{\sigma}s \notin T$, then x_i is coded using P_s and the average code length of x_i is $H(P_s)$. This will happen $w(\hat{\sigma}s)$ times out of $w(s)$ times we visit s.

2. If $\hat{\sigma}s$ is present in T, then x_i is coded according to distribution of some node in $T_{\hat{\sigma}s}$, and the average code length will be the entropy of the distribution in that node. This will also happen $\frac{w(\hat{\sigma}s)}{w(s)}$ out of times we visit s.

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

Thus the entropy of T_s satisfies the recursive definition:

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

And the code length of data "passing through" s is $w(s) \cdot H(T_s)$.

Summing this altogether we get the description length:

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

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 (see Fig. 3.2):

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.

Chapter 4

Unsupervised Learning and Clustering Algorithms

4.1 Mixture Models

Up to now we have talked about modeling and learning of single source sources. Now we assume that our source G is a collection of sources $G_1, .., G_k$. We also assume that G has a distribution $P(G_j)$ over its sub-sources. The data points are generated by G in the following way: first we choose a sub-source G_j according to $P(G_j)$ and then we let G_j to generate a new data point. The resulting *mixture distribution* is given by:

$$p(x) = \sum_{j=1}^{k} P(G_j) \cdot p(x|G_j)$$

In this chapter we are going to survey the existing methods for learning of mixture distributions. Learning of mixture distribution includes:

1. Inferring the number of mixture components k.
2. Learning each of the sources (finding $p(x|G_j)$).
3. Learning the distribution over the sub-sources $P(G_j)$.
4. For each data point determining, which sub-source has most probably generated it.

A set of points that were generated by the same sub-source will be called a *cluster*. Thus, the procedure of separation of $\mathbf{x} = x_1, .., x_n$ into subsets that most probably emerged from the same sub-source is called *clustering*.

Note, that sometimes two different mixtures may provide the same output distribution $p(x)$. For example, for a mixture of two coins, where at each step we randomly with probability $(0.5, 0.5)$ choose one coin, flip it and write down the result, the output distribution of a pair (mixture) $\{(0.4, 0.6), (0.6, 0.4)\}$ and a pair $\{(0.3, 0.7), (0.7, 0.3)\}$ will be the same. So, by looking at the result we will not be able to distinguish between the two. This problem is called *unidentifiability* of a mixture (see [YS68, DHS01]). While we will not enter into discussion of this

question here, it should be mentioned that the algorithm described in Sec. 4.3 will return the minimal mixture that describes the sample best as the most probable one, as will be described there (e.g. in the given example it will be the mixture of one coin with $(0.5, 0.5)$ distribution).

4.2 Expectation Maximization

The first procedure for learning of mixture distributions we are going to describe here is the *Expectation Maximization (EM)* algorithm, suggested in [DLR77] (see also [MK97] for an overview). EM is a general framework for learning from *incomplete data*. In the case of learning of mixture distributions, the incomplete (or *missing*) data is the correspondence between the points $x_1, .., x_n$ and the sub-sources they emerged from. Given that correspondence we could learn each source separately using some technique from Ch. 2.

The EM algorithm aims to find the mixture that will maximize the likelihood $\mathcal{L}(\mathbf{x}|\theta)$ of the data. θ represents here the complete set of parameters of the mixture. It should be noted, that in most non-trivial cases EM is not guaranteed to find the global optimum, but only a local one.

In the context of clustering the EM algorithm works in the following way. It is assumed that the number of mixture components k is known. We denote by $\{x_i \in G_j\}$ the event that x_i was generated by the sub-source G_j. The events $\{x_i \in G_j\}$ are our hidden parameters. The maximization of the likelihood of the observation $\mathcal{L}(\mathbf{x}|\theta)$ is equivalent to maximization of the log likelihood:

$$\log \mathcal{L}(\mathbf{x}|\theta) = \log p(x_1, .., x_n|\theta) = \sum_{i=1}^{n} \log p(x_i|\theta)$$

The last equality follows from the independence assumption we return to in this chapter.

In order to ease the notations we will talk about maximization of $\log p(x|\theta)$. The reader may easily see that the summation over i, $\sum_{i=1}^{n}$, may be added before each term in the equations we are going to have here.

We can write:

$$p(x, x \in G_j|\theta) = p(x|\theta) \cdot p(x \in G_j|x, \theta) \tag{4.1}$$

We add one more simplification and write: $p(x \in G_j|x, \theta) = p(G_j|x)$. (Conditioning on θ, while always present, is omitted for better readability and the event $\{x \in G_j\}$ is written simply as G_j.) By taking $p(G_j|x)$ to the second side of (4.1) and applying log we get:

$$\log p(x|\theta) = \log p(x, G_j|\theta) - \log p(G_j|x) \tag{4.2}$$

Note also, that

$$p(x, G_j|\theta) = p(G_j|\theta) \cdot p(x|G_j, \theta) = P(G_j) \cdot p(x|G_j) \tag{4.3}$$

We may always write:

$$\log p(x|\theta) = \log p(x|\theta) \sum_j p(G'_j|x) = E_{p(G'_j|x)} \log p(x|\theta)$$

Applying the same to the second side of (4.2) and substituting (4.3) we get:

$$E_{p(G'_j|x)} \log p(x|\theta) = E_{p(G'_j|x)} \log P(G_j)p(x|G_j) - E_{p(G'_j|x)} \log p(G_j|x)$$

Put an attention that we took the expectation over some other mixture θ' with sub-sources G'_j.

If we have an algorithm to find θ' such that

$$E_{p(G'_j|x)} \log P(G'_j)p(x|G'_j) \geq E_{p(G'_j|x)} \log P(G_j)p(x|G_j) \tag{4.4}$$

then:

$$\begin{aligned} &\log p(x|\theta') - \log p(x|\theta) \\ &= E_{p(G'_j|x)} \log P(G'_j)p(x|G'_j) - E_{p(G'_j|x)} \log P(G_j)p(x|G_j) \\ &+ E_{p(G'_j|x)} \log p(G'_j|x) - E_{p(G'_j|x)} \log p(G_j|x) \\ &\geq D_{KL}[p(G'_j|x)\|p(G_j|x)] \geq 0 \end{aligned}$$

Indeed, the idea of EM is to start with some initial guess θ^0 and then iteratively fix θ^m and find θ^{m+1} such that (4.4) holds when we think of θ^m as θ and θ^{m+1} as θ'. Due to monotonic increase of $\log p(x|\theta)$ the algorithm is ensured to converge to some local optimum.

All we left to do is to define $p(G_j|x)$ in order to be able to compute and maximize $E_{p(G'_j|x)} \log P(G'_j)p(x|G'_j)$. There are a number of ways of defining $p(G_j|x)$. If

$$p(G_j|x) = \begin{cases} 1 & \text{if } \forall j' \; p(x|G_j) \geq p(x|G_{j'}) \\ 0 & \text{otherwise} \end{cases}$$

we get the *k-means* clustering algorithm. And if

$$p(G_j|x) = \frac{P(G_j)p(x|G_j)^\beta}{\sum_{j'} P(G_{j'})p(x|G_{j'})^\beta}$$

we get the *fuzzy k-means* clustering algorithm (see [DHS01]). Note, that for $\beta = 1$ the above definition coincides with the Bayesian definition of $p(G_j|x)$.

We will return to the question of definition of $p(G_j|x)$ in the next section.

4.3 Clustering from the Information Theoretic Point of View

The drawback of the EM algorithm is that it "gets stuck" in the first local maximum, closest to the initial guess θ^0. It is common to run the EM multiple times with random initial starting points to cope with this problem, but this may still give little improvement for likelihood functions $p(\mathbf{x}|\theta)$ riddled with local maxima (as a function of θ). In addition, EM does not solve the problem of determining the number of components in the generating mixture.

What we would like to do is to work at increasing levels of resolution. When working at low resolution we will look on a smoothed likelihood function that will have much less local maxima - hopefully just one. Finding this maximum (with EM-like procedure) will bring us to the highest region of the likelihood function. By slowly increasing the resolution we will "get up" in that area and, with a bit of luck, get to the real optimum of the likelihood function. Of course, we are not provided with any guarantees to find the global optimum, but many local optima will be automatically avoided, taking us to qualitatively new levels of solutions.

In this section we are primarily focusing on the negative log likelihood function $-\log p(\mathbf{x}|\theta)$. Note, that maximization of $p(\mathbf{x}|\theta)$ is equivalent to minimization of $-\log p(\mathbf{x}|\theta)$. Fig. 4.1 shows

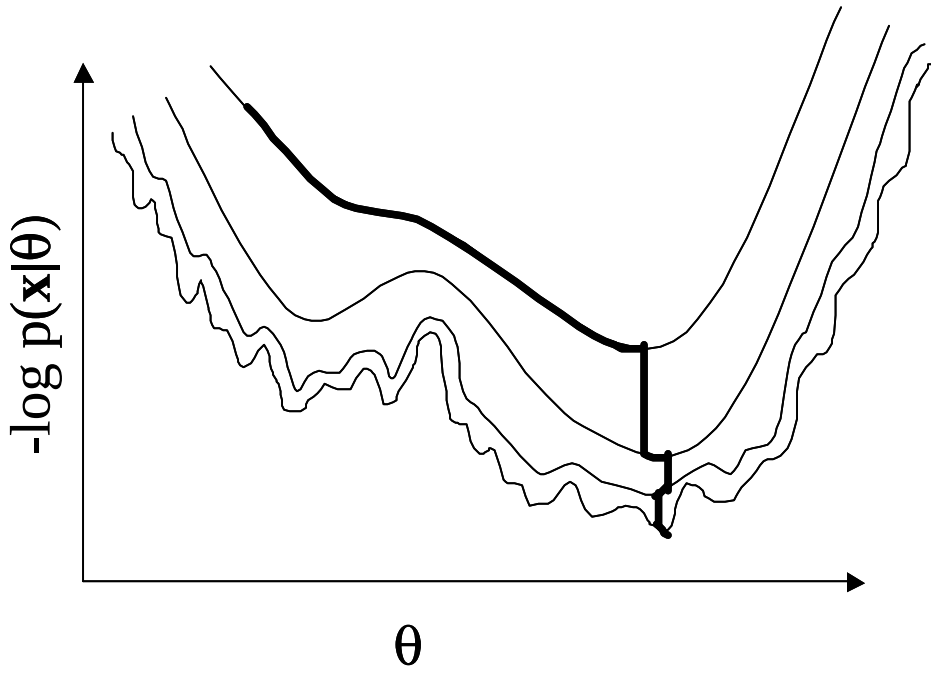


Figure 4.1: **Working at increasing levels of resolution - an illustrative example.** The bottom curve is the negative log likelihood function $-\log p(\mathbf{x}|\theta)$ as a function of θ. Curves above it are the negative log likelihood function at decreasing levels of resolution (smoothing of the original one). The bold line is an imaginary search path that starts at a random θ at low resolution and goes down the negative log likelihood function while increasing the resolution as it gets to a local minima at the current level.

an illustrative example of our attitude to minimization of $-\log p(\mathbf{x}|\theta)$, and Fig. 4.2 gives an intuitive example of potential vulnerability of the method.

In what is following we are going to describe the *deterministic annealing* framework that realizes the described attitude to clustering. But before this we review some more basics from the Information Theory.

4.3.1 Rate Distortion Theory

The idea of rate distortion theory was introduced by Shanon in his original paper [Sha48]. Rate distortion theory deals with the question of optimal quantization of random variables. Unlike in Ch. 2 and 3, where we dealt with *loss-less compression*, this time we are allowed to (slightly) distort the coded information in the meaning that the output of the decoder may be not exactly the input of the encoder. Therefore quantization is also called *lossy compression*. In this problem setting we have a vector of independent identically distributed (i.i.d) random variables $\mathbf{x} = x_1, .., x_n \in \mathcal{X}^n$, generated by a stochastic source. The goal is to code (represent) the sequence with an estimate $\hat{\mathbf{x}} = \hat{x}_1, .., \hat{x}_n \in \hat{\mathcal{X}}^n$ that will optimize a compound criterion on the size of $\hat{\mathcal{X}}$ and the quality of subsequent decoding of $\hat{\mathbf{x}}$ (or simply the distance between $\mathbf{x}$ and

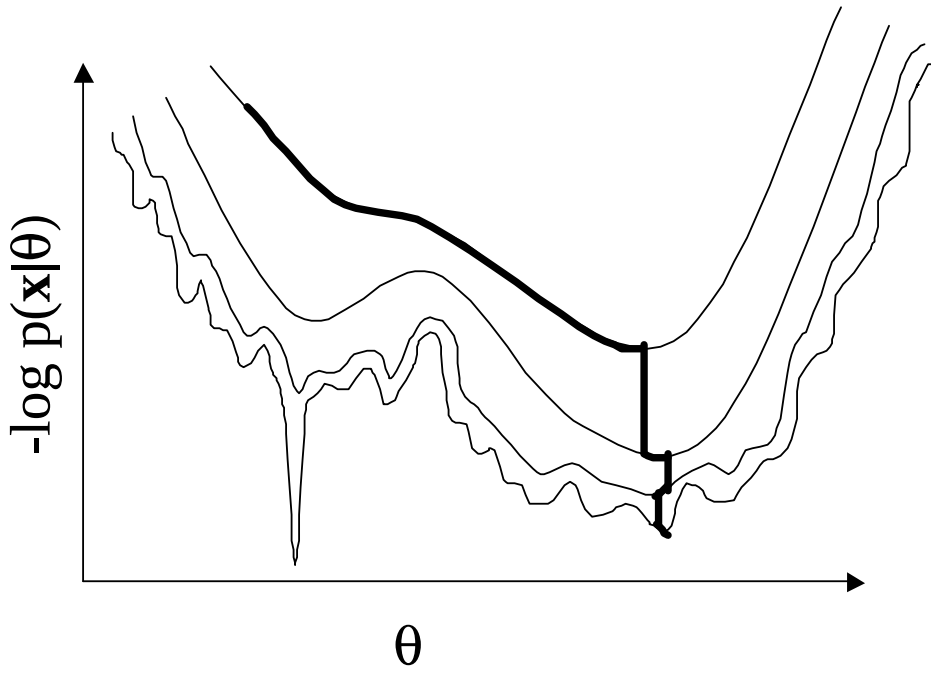


Figure 4.2: **Working at increasing levels of resolution - potential vulnerability.** Narrow deep minima of the negative log likelihood function may be missed up when working at increasing levels of resolution, as may be intuitively seen in this example.

$\hat{\mathbf{x}}$). This criterion will be given immediately after a sequence of definitions that form the rate distortion theory.

Definition 4.1 *A* distortion function *or* distortion measure *is a mapping* $d : \mathcal{X} \times \hat{\mathcal{X}} \to \Re^+$. *The distortion* $d(x, \hat{x})$ *is a measure of the cost of representing* $x \in \mathcal{X}$ *by* $\hat{x} \in \hat{\mathcal{X}}$.

Definition 4.2 *The* distortion between sequences $\mathbf{x} = x_1, .., x_n$ *and* $\hat{\mathbf{x}} = \hat{x}_1, .., \hat{x}_n$ *is defined by* $d(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \hat{x}_i)$.

Definition 4.3 *A* $(2^{nR}, n)$ rate distortion code *consists of an encoding function*

$$f_n : \mathcal{X}^n \to \{1, .., 2^{nR}\}$$

and a decoding (reproduction) function

$$g_n : \{1, .., 2^{nR}\} \to \hat{\mathcal{X}}^n$$

The distortion associated with $(2^{nR}, n)$ *code is defined as:*

$$D = E_{p(\mathbf{X})} d(\mathbf{X}, g_n(f_n(\mathbf{X})))$$

Definition 4.4 *A rate distortion pair* (R, D) *is said to be* achievable *if there exists a sequence of* $(2^{nR}, n)$ *rate distortion codes* (f_n, g_n) *with*
$\lim_{n\to\infty} E_{p(\mathbf{X})} d(\mathbf{X}, g_n(f_n(\mathbf{X}))) \leq D$.

Definition 4.5 *The* rate distortion region *for a source is the closure of the set of achievable rate distortion pairs* (R, D).

Definition 4.6 *The* rate distortion function $R(D)$ *is the infimum of rates* R *such that* (R, D) *is in the rate distortion region of the source for a given distortion* D.

We also define:

$$\langle d \rangle = \sum_{x\in\mathcal{X}, \hat{x}\in\hat{\mathcal{X}}} p(x)p(\hat{x}|x)d(x, \hat{x})$$

In [CT91] you may find a proof that:

$$R(D) = \min_{p(\hat{x}|x):\langle d\rangle \leq D} I(X, \hat{X}) \tag{4.5}$$

Here $\hat{\mathcal{X}}$ is fixed. The equation says that the optimal assignment probabilities $p(\hat{x}|x)$ are those that minimize the mutual information between x and $\hat{x}$, while keeping on the desired level of distortion. Such assignment satisfies the distortion constraint on $p(\hat{x}|x)$, but makes no other assumptions (or constrains) on the relation between x and $\hat{x}$, and thus it is the most probable one.

The optimal assignment probabilities may be found with the Blahut-Arimoto (BA) alternating minimization procedure. The procedure starts with an initial guess of $p(\hat{x})$ and β and then iteratively repeats the calculations:

$$p(\hat{x}|x) = \frac{p(\hat{x})e^{-\beta d(x,\hat{x})}}{\sum_{\hat{x}'} p(\hat{x}')e^{-\beta d(x,\hat{x}')}} \tag{4.6}$$

$$p(\hat{x}) = \sum_{x\in\mathcal{X}} p(x)p(\hat{x}|x) \tag{4.7}$$

until their convergence.

This algorithm was suggested by Blahut [Bla72] and Arimoto [Ari72] and proved to converge to the rate distortion function by Csiszar [Csi74]. The distortion constraint D is replaced in the calculations by corresponding Lagrange multiplier β. By choosing β appropriately we can sweep out the $R(D)$ curve. See [CT91, Ch. 13] for a more detailed discussion of the algorithm and the theory in whole.

4.3.2 Rate Distortion and Clustering

Note, that what we have just done was a partition of $\mathcal{X}$ into $|\hat{\mathcal{X}}|$ clusters with "soft" assignment of each point $x \in \mathcal{X}$ to each cluster $\hat{x} \in \hat{\mathcal{X}}$: $p(\hat{x}|x) = p(x \in \hat{x})$. The Lagrange multiplier β plays the role of resolution parameter - for small β the actual distance $d(x, \hat{x})$ has low influence on $p(\hat{x}|x)$, while for β tending to infinity we converge to the hard partition of the data, the so called "winner takes all" (see definition of the assignment probabilities in (4.6)).

Also, note that $\hat{\mathcal{X}}$ need not necessarily be a subspace of $\mathcal{X}$. We may take $\hat{\mathcal{X}}$ to be a set of data generating models over $\mathcal{X}$, and define $d(x,\hat{x})$ to be $-\log p_{\hat{x}}(x)$. For example, if we work with points in $\Re^n$, $\hat{\mathcal{X}}$ may be a collection of Gaussians. The reason for taking $-\log p_{\hat{x}}(x)$ as our distance measure is that if we measure a distance of a set of i.i.d. data points from a source, we will get: $d(x_1,..,x_n;\hat{x}) = -\log p_{\hat{x}}(x_1,..,x_n) = -\log\prod_i p_{\hat{x}}(x_i) = \sum_i -\log p_{\hat{x}}(x_i) = \sum_i d(x_i,\hat{x}_i)$. I.e. the distances are summing up as we would like them to behave.

Now assume that we have an algorithm to find the most likely model for a weighted data set. The algorithm finds:

$$\hat{x}^* = \arg\min_{\hat{x}'} \sum_i p(\hat{x}|x_i) d(x_i,\hat{x}')$$

where $p(\hat{x}|x_i)$ are the weights. (Put attention that the weighting is over fixed weights $p(\hat{x}|x_i)$, and that learning procedure returns a new model $\hat{x}^*$.) Then for our distance measure:

$$\hat{x}^* = \arg\min_{\hat{x}'} \sum_i p(\hat{x}|x_i) d(x_i,\hat{x}') = \arg\min_{\hat{x}'} \sum_i p(\hat{x}|x_i)\cdot(-\log p(x_i|\hat{x}'))$$

$$= \arg\max_{\hat{x}'} \sum_i p(\hat{x}|x_i)\log p(x_i|\hat{x}')$$

Which gives us an inequality:

$$\sum_i p(\hat{x}|x_i)\log p(x_i|\hat{x}^*) \geq \sum_i p(\hat{x}|x_i)\log p(x|\hat{x}) \tag{4.8}$$

Recalling from the previous section that: $\log p(x|\theta) = \log p(\hat{x})p(x|\hat{x}) - \log p(\hat{x}|x) = \log p(\hat{x}) + \log p(x|\hat{x}) - \log p(\hat{x}|x)$, and observing that at this point $p(\hat{x}^*) = p(\hat{x})$ we get:

$$\sum_i [\log p(x_i|\theta^*) - \log p(x_i|\theta)]$$

$$= \sum_i [\log p(\hat{x}^*) + \log p(x_i|\hat{x}^*) - \log p(\hat{x}^*|x_i) - \log p(\hat{x}) - \log p(x_i|\hat{x}) + \log p(\hat{x}|x_i)]$$

$$= \sum_i \sum_{\hat{x}} [p(\hat{x}|x_i)(\log p(x_i|\hat{x}^*) - \log p(x_i|\hat{x})) + p(\hat{x}|x_i)(\log p(\hat{x}|x_i) - \log p(\hat{x}^*|x_i))]$$

$$= \sum_{\hat{x}} [\sum_i p(\hat{x}|x_i)(\log p(x_i|\hat{x}^*) - \log p(x_i|\hat{x}))] + \sum_i D_{KL}[p(\hat{x}|x_i)\|p(\hat{x}^*|x_i)] \geq 0$$

Thus we have proved a monotonic increase in the likelihood function. This means that if we start from some initial guess of $\{\hat{x}_j\}$ and iteratively fix the set we have and find a new set $\{\hat{x}_j^*\}$ that will satisfy (4.8) we will converge to some local optimum of the likelihood function, like in EM. The essence of rate distortion based clustering is that we do not try to find the most likely model for the data, but rather optimize the rate distortion function, while this time we are allowed to manipulate not only with the assignment probabilities $p(\hat{x}|x)$, but also with the models (or centroids) $\hat{x}$ themselves. This way the expression we try to optimize is:

$$\min_{\{\hat{x}\},p(\hat{x}|x):\langle d\rangle\leq D} I(X,\hat{\mathcal{X}})$$

which makes us to define the assignment probabilities $p(\hat{x}|x)$ by (4.6).

There is one important point in rate distortion based clustering we want to mention here. For each level of distortion D there is a finite number of models $K(D)$ that are required in order to describe $\mathbf{x}$ at distortion not greater than D. (Definitely $K(D) \leq n$, where n is the size of our sample.) If we start our clustering procedure with $k > K(D)$ models, at the end of the clustering with high probability some of the models will unite together (coincide) or remain with no data (no data points will be assigned to those models), leaving us with $K(D)$ "*effective*" - distinct and non-empty models. This happens due to the following reason. Our optimization goal may be written as:

$$R(D) = \min_{\hat{x}, p(\hat{x}|x):\langle d\rangle \leq D} I(X,\hat{X}) = \min H(X) - H(X|\hat{X}) = H(X) - \max H(X|\hat{X})$$

$$= H(X) - \max \sum_{x\in\mathcal{X}, \hat{x}\in\hat{\mathcal{X}}} -p(x|\hat{x}) \log p(x|\hat{x}) = H(X) + \min \sum_{x,\hat{x}} p(x|\hat{x}) \log p(x|\hat{x})$$

Due to the concavity of the $\alpha \log \alpha$ function by Jensen inequality:

$$(p(x|\hat{x}_1) + p(x|\hat{x}_2)) \log(p(x|\hat{x}_1) + p(x|\hat{x}_2)) \leq p(x|\hat{x}_1) \log p(x|\hat{x}_1) + p(x|\hat{x}_2) \log p(x|\hat{x}_2)$$

And equality holds iff $p(x|\hat{x}_1) = p(x|\hat{x}_2)$ or one of $p(x|\hat{x}_1), p(x|\hat{x}_2)$ is zero. Thus, unification or elimination of clusters ($\hat{x}_1$ and $\hat{x}_2$) reduces the mutual information $I(X,\hat{X})$ making solutions with lower number of distinct clusters more preferable whenever those solutions achieve the required level of distortion. A more detailed discussion of this phenomenon may be found in [Ros98].

4.3.3 Hierarchical Clustering through Deterministic Annealing

Deterministic annealing (DA) is a general framework that enables clustering at increasing levels of resolution as was described at the beginning of this section. DA does not require information about the number of models in a mixture, but rather finds the most likely one, thus solving the problem we could not solve with regular EM. DA teaches us to act in the following way.

We start with low initial value of the resolution parameter β and train a single model using all the data we have. Note, that low value of β corresponds to high level of permitted distortion D, thus for β small enough one cluster will suffice to describe $\mathbf{x}$ at the required level of distortion.

Then we create two copies of our model and perform random, usually asymmetric, perturbations on each of the copies. We will call this operation *split*.

With the two models we got after split we run the rate distortion based clustering as described in the previous subsection. As mentioned there, at the end of the clustering we may remain with two different, two identical or one empty and one "full" models.

The last two cases mean that we have reached the essential number of models, required to describe $\mathbf{x}$ at the current level of distortion $D(\beta)$, and there is nothing to further look for at the current value of β. Therefore we unite all coinciding and remove all empty models, increase β, split all the models we have at hand and repeat clustering with the new set of models we got.

If after clustering the number of effective models did increase, this means that we possibly have not reached yet the limit required at the current level of resolution. Therefore we unite all the coinciding models and eliminate all empty models as previously, but return to clustering without increasing β.

Clusters that do not split up over long ranges of β are *stable clusters* (after we split the cluster representative model and run the clustering procedure, the two copies either return to be together or one of the models is "pushed out" and the second one takes all the samples of the parent model). Stable clusters carry important information on the statistical structure of our sample, and in particular on the underlying mixture model.

Note, that the history of splits forms a tree hierarchy of models for our sample. We start with a single model at the root of that tree and repeatedly split our model (and correspondingly our sample), optionally till the limit when each point in the sample is represented by a separate cluster. This way of clustering is called *hierarchical top-down* or *divisive clustering*, as opposed to *hierarchical bottom-up* or *agglomerative clustering* where we start from the limit of taking each point to be a separate cluster and repeatedly unite clusters together until we get one big cluster for the whole data (see [DHS01] for an example of such algorithm). Obviously, the approach described here is highly preferred on the agglomerative clustering in the cases when we have a large sample that splits into few large clusters we are interesting to find out.

To avoid duplications, a pseudocode for the algorithms described in this chapter will be given in the next one together with our algorithm. A more detailed discussion of DA may be found in [Ros98].

Chapter 5

Unsupervised Sequence Segmentation

5.1 Problem Formulation

At this point we are finally ready to discuss the main issue of this work - the unsupervised sequence segmentation problem. Our input is a string $\bar{x} = x_1..x_n$ that was generated by a mixture $\{T_j^*\}_{j=1}^{k^*}$ of VMM models in the following way. At each particular point x_i of $\bar{x}$ only one source was active. Each source was allowed to generate a number of consequent symbols and only then, at a random time i, it was switched by another source that was possibly already active in the preceding segments of $\bar{x}$. Our assumptions are:

1. We do not know the number of sources k^*.
2. We do not know the sources $\{T_j^*\}$.
3. We do not know at which places of $\bar{x}$ which source was active.
4. We assume that when some source $T_{\mathbf{j}}^*$ is activated, it is activated for significantly long period of time, so that the generated segment will be "long enough". Long enough means that if we had the models $\{T_j^*\}_{j=1}^{k^*}$, we could say with very high confidence that the subsequence was generated by $T_{\mathbf{j}}$ and not by any other source (we will call this property *distinguishability* of the model on the segment).
5. We assume that the total length of the segments generated by each source is sufficient to build a "good" model of that source. A model is good if using that model we can distinguish the corresponding source from all other sources on each separate segment it generated.

It is easy to see the correlation between the assumptions 4 and 5: if we have more data we can distinguish between the same sources at higher alternation rates.

Our goal is:

1. To infer the most likely number of mixture components k.
2. To build a model T_j for each component of the mixture.

3. To partition the sequence into segments, when each segment was (most likely) generated by a single mixture component and identify that component.

Of course, the main goal is to get the segmentation as close as possible to the original one.

5.2 Unsupervised Sequence Segmentation Algorithm

We approach the problem described above with our new algorithm for unsupervised sequence segmentation, first presented in [SBT01b] . The algorithm is based on the observation that with PSTs we have:

1. Models that induce probability distribution over sub-strings of $\bar{x}$.
2. An algorithm for training a new model given a string $\bar{x}$ and a vector of weights $\bar{w}$ (see Sec. 3.2.3).

Thus we may use the deterministic annealing framework described in Sec. 4.3 to find the number of sources that generated $\bar{x}$, to model the sources themselves and to obtain the most likely segmentation of $\bar{x}$ that will emerge from clustering of the elements of $\bar{x}$, x_i-s, taken in their context in $\bar{x}$.

We define the distance between a symbol x_i and a PST model T_j to be negative log likelihood T_j induces 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. Note that in order to explore the structure of a source T_j we need it generate continuous segments. If we were switching sources too frequently, we would not see the time dependencies $P_{T_j}(x_i|x_1..x_{i-1})$ of each specific source, but rather get an unidentifiable mixture. With the smoothing window x_i-s close in space (i.e. with small difference in i) will with high probability be close to the same model T_j since their windows will significantly overlap.

It should be mentioned that taking a window centered on x_i instead, for example taking a window of a form $x_{i-(2M+1)}..x_i$, on practice improves the performance at least by a factor of two in the sense that we may distinguish between the same sources when they alternate twice more frequently. The point of transition is also better determined with a symmetric window; with asymmetric one it is shifted to the side opposite to the "mass center" of the window.

Having defined the distance between a symbol and a model we may find the optimal assignment probabilities $P(T_j|x_i)$ (when x_i is viewed in its context in $\bar{x}$) for a *fixed* set of PST models. We denote the set by $\mathcal{T} \equiv \{T_j\}_{j=1}^k$. Of course, the optimality is relatively to the distance measure we have chosen, since we optimize (4.5). The assignment probabilities are obtained through the Blahut-Arimoto alternating minimization procedure described in Sec. 4.3.1 - see Fig. 5.1 for a pseudocode. Here $\frac{1}{n}\sum_{i=1}^n$ is an empirical approximation of the expectation $\sum_{x\in\mathcal{X}} p(x)$ in (4.7).

As in Sec. 4.3 we improve our clustering by allowing *retraining* of the PST models (update of the centroids of the clusters); the number of models is still hold fixed. We define $w_i^j \equiv P(T_j|x_i)$, thus $\bar{w}^j = w_1^j..w_n^j$ is a vector of weights associated with $\bar{x}$ and a model T_j. For model retraining

Blahut-Arimoto($P(T_*)$, 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. $\forall j: \quad P(T_j) = \frac{1}{n}\sum_{i=1}^{n} P(T_j|x_i)$

Figure 5.1: **The Blahut-Arimoto algorithm.**

Soft_Clustering($\mathcal{T}$, $P(T_*)$, β)

Repeat until convergence:

1. Blahut-Arimoto($P(T_*)$, β)
2. $\forall j: \; T_j = \text{Learn\_PST}(\bar{x}, \bar{w}^j)$

Figure 5.2: **Soft_Clustering procedure.**

we use the MDL-based algorithm for PST learning described in Sec. 3.2.3. Clustering is done through yet another alternating minimization procedure, as described in Sec. 4.3.2. The soft clustering starts with some initial guess of the models set $\mathcal{T}$ and a prior distribution over $\mathcal{T}$, $P(T_*)$. Then we alternatively fix the models and find the association probability vectors $\{\bar{w}^j\}_{j=1}^{k}$, delete all the models we have and train a new set of models (of the same size) using the association probability vectors we got (see Fig. 5.2).

We remind that as described in Sec. 4.3.2, for each level of resolution β there is a finite number of models $K(\beta)$, required to described $\bar{x}$ at distortion bounded by $D(\beta)$. If we start our soft clustering with $k > K$ models, at the end of it we will remain with only K active models. While in the case of Gaussians clustering the number of effective models decreases due to unification of clusters, in sequence segmentation the more frequent event is a disappearance of a cluster (cluster representative model remains with no data assigned to it). This is caused by the usage of the averaging windows in the definition of our distance measure. Models having small amounts of data slowly loose their "weight" in all the windows and the competement over the windows pushes them out.

The only thing left on our way to embedding of our clustering algorithm in the deterministic annealing framework is a definition of the procedure for splitting of the PSTs in $\mathcal{T}$. This is done in a rather simple manner (see Fig. 5.3). 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.

Split_PSTs($\mathcal{T}$, $P(T_*)$)

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 renormalize 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. $P(T_{j_1}) = \frac{1}{2}P(T_j),\ P(T_{j_2}) = \frac{1}{2}P(T_j)$

Figure 5.3: **The Split procedure.**

Now we are finally ready to outline the complete algorithm. We start with $\mathcal{T}$ including a single "average" PST T_0 that is trained on the whole sequence $\bar{x}$ with $w(x_i) = 1$ for all i. We then pick an initial value of β, split $\mathcal{T}$ and proceed with the soft clustering procedure that is initialized with the two models we got after split. We then split $\mathcal{T}$ again and repeat. If a model is found to have lost all its data it is eliminated [1]. When the number of effective models stops increasing we increase β and repeat the whole process.

We continue to increase β till the limit when the clusters become just one window size. This corresponds to the limit when each point is a separate cluster since the window size is the maximal resolution we can achieve.

See Fig. 5.5 for a pseudocode of the algorithm and Fig. 5.4 for a schematic description.

5.3 Remarks

Algorithm's limitations

As already mentioned in Sec. 5.1, the input string should have enough data to build reasonable models for each of the sources. The alternation between sources should not be faster than the one we can distinguish with the models if were built from the data in a supervised fashion. I.e. if someone was giving us the true segmentation of the sequence and we were training a set of models, each on its segments of the data, those models would be well distinguishable on the segments of the true segmentation.

It should be noted that the above limitation is an inherent limitation of any algorithm that attempt sequence segmentation. Being unsupervised one, our algorithm requires a bit slower

[1] The effect of unification of models is very infrequent in the process of sequence segmentation and therefore was not treated.

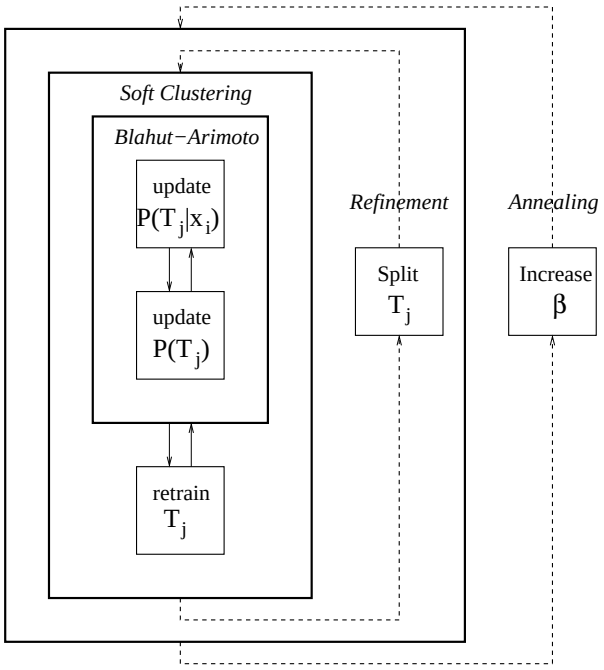


Figure 5.4: **A schematic description of the algorithm**.

The segmentation algorithm:

Initialization:

For all i, $w^0(x_i) = 1$
$T_0 = \text{Learn\_PST}(\bar{x}, \bar{w}^0)$
$\mathcal{T} = \{T_0\}$, $P(T_0) = 1$
$\beta = \beta_0$, $k_{prev} = 0$

Annealing loop:

While $|\mathcal{T}| < \frac{n}{2M+1}$

1. While $|\mathcal{T}| > k_{prev}$
   (a) $k_{prev} = |\mathcal{T}|$
   (b) Split_PSTs($\mathcal{T}$, $P(T_*)$)
   (c) Soft_Clustering($\mathcal{T}$, $P(T_*)$, β)
   (d) Remove all T_j such that $P(T_j) = 0$ from $\mathcal{T}$.
2. Increase β

Figure 5.5: **Unsupervised Sequence Segmentation algorithm.**

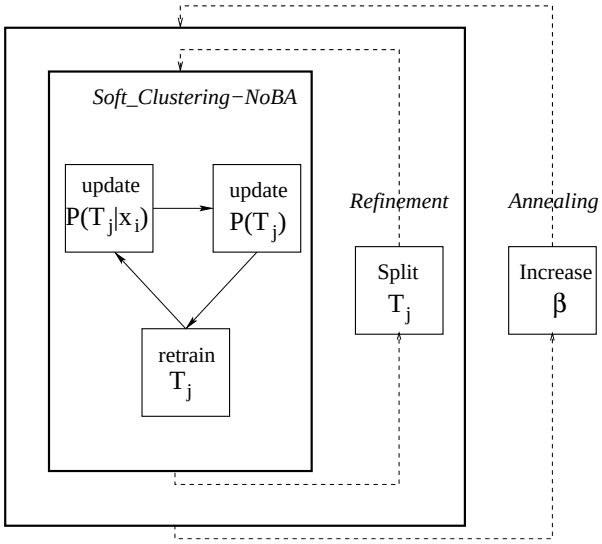


Figure 5.6: **A schematic description of the algorithm based on Soft_Clustering-NoBA procedure.** Compare this scheme to the one in Fig. 5.4.

alternation rates - about two or three times longer segments than the distinguishable ones, but that is quite reasonable.

There is one more limitation that is specific to our algorithm - the averaging window size. At the moment the size of the window is an external parameter and we can not alter switching rates that are faster than one switch per window. We want to note, that while large windows diminish our ability to distinguish between quickly alternating sources, they provide us with a more confident segmentation (i.e. we have a significant difference in $P(T_j|x_i)$ of the model that is active on a segment and all the rest models). For too small windows the segmentation may appear to be too noisy to be of any usefulness, as well as the segmentation process may "get lost" and suggest some occasional and meaningless local minima segmentation, especially if the input data is noisy. Thus the correct choice of the size of the averaging window is a hard problem by itself we are working on now. Currently the size of the window is manually chosen depending on the kind of the input data.

Usage of the BA

As appeared in practical applications we had, it is better to make just a single pass in the BA loop instead of running it till convergence. This means that our soft clustering procedure does not use the BA, but rather makes the calculation from the BA loop a single time between two consequent retrains of $\mathcal{T}$, as shown in Fig. 5.7.

This happens due to the following reason. When we do not run the BA loop till convergence we spend less time looking for the optimal assignment probabilities $P(T_j|x_i)$ and more in training new sets of models $\mathcal{T}$. Since $\mathcal{T}$ is the determining component of the mixture, a more extensive search over the space of possible $\mathcal{T}$-s is beneficial - we give our algorithm the possibility to correct the choice of $\mathcal{T}$ *while* it looks for the optimal assignment probabilities. See Fig. 5.6 for

Soft_Clustering-NoBA($\mathcal{T}$, $P(T_*)$, β)

Repeat until convergence:

1. $\forall i, j: \; w^j(x_i) = P(T_j|x_i) = \frac{P(T_j)e^{-\beta d(x_i,T_j)}}{\sum_{\alpha=1}^{k} P(T_\alpha)e^{-\beta d(x_i,T_\alpha)}}$
2. $\forall j: \quad P(T_j) = \frac{1}{n}\sum_{i=1}^{n} P(T_j|x_i)$
3. $\forall j: \quad T_j = \text{Learn\_PST}(\bar{x}, \bar{w}^j)$

Figure 5.7: **Soft_Clustering-NoBA procedure**

a schematic description of this new version of the algorithm. The "no-BA" modification of the algorithm was successfully used in our [BSMT01] and [SBT01a] works.

Convergence of the algorithm

Unfortunately at the moment we do not have a proof of convergence of our algorithm. Actually, the soft clustering procedure sometimes enters small "oscillations" around the point of convergence (very small amounts of data pass from one model to another and backwards) and does not converge in the strict sense. This happens because a stronger model can "steel" some data from a weaker one using the smoothing window. But this weakens the strong model (since the data stolen has different statistical structure and adds noise) and at the next iteration it looses the stolen data back. The convergence may be forced by external control (like limiting the number of iterations, enlarging β inside the clustering process or entering small perturbations if the process does not converge for a long time), but we see this as an "inelegant" solution. Hopefully the problem will be solved with the improvements of the algorithm suggested in Ch. 7.

Chapter 6

Applications

6.1 Multilingual Texts Segmentation

The first application described here may look a bit artificial (though we do not deny a possibility of finding a similar problem in the real life), but it is very useful for general understanding of our algorithm, since the data used is well explored and intuitively feasible.

In this 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 is 150000 letters (30000 from each language).

We made several independent runs of our algorithm, when based on the Soft_Clustering-NoBA procedure (see Fig. 5.6). In every run after about 2000 accumulated innermost (Soft_Clustering-NoBA loop) 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. 6.1 and 6.2 for a typical example). 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 (see Fig. 6.4). Also, in most runs linguistically similar languages (English and German; French and Italian) separated at later stages of the segmentation process, suggesting a hierarchical structure over the discovered data sources (Fig. 6.3 gives an example).

6.1.1 The Clustering Process

To give a better understanding of our algorithm we turn to demonstrate the details of the development (or evolution) of the clustering process on the multilingual text example. We start with discussion of the segmentation algorithm based on the Soft_Clustering-NoBA procedure (Fig. 5.6), since this one was used to obtain all the results presented in this work. We then compare it with the algorithm based on Blahut-Arimoto procedure, when we run it till the convergence (Fig. 5.4), and show the drawbacks of such approach.

In Fig. 6.3 we depict the probabilities of each of the models in $\mathcal{T}$, as calculated in step 2. of the Soft_Clustering-NoBA procedure (Fig. 5.7), as a function of cumulative number of this iteration. The values of β and points of its increments (after convergence of the number of models at the current value of β) are written below the x axis. "Falls down" and subsequent

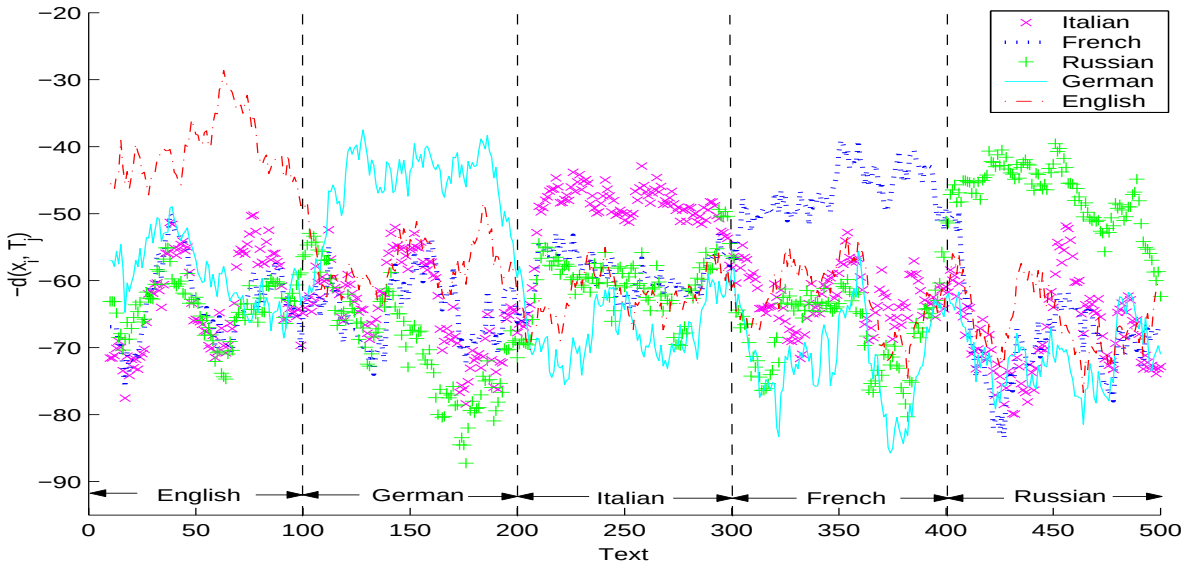


Figure 6.1: **Multilingual text segmentation**. The graph shows $-d(x_i, T_j)$ for the first 500 letters of text, for all $T_j \in \{T_{English}, T_{German}, T_{Italian}, T_{French}, T_{Russian}\}$. The true segmentation appears at the bottom of the graph.

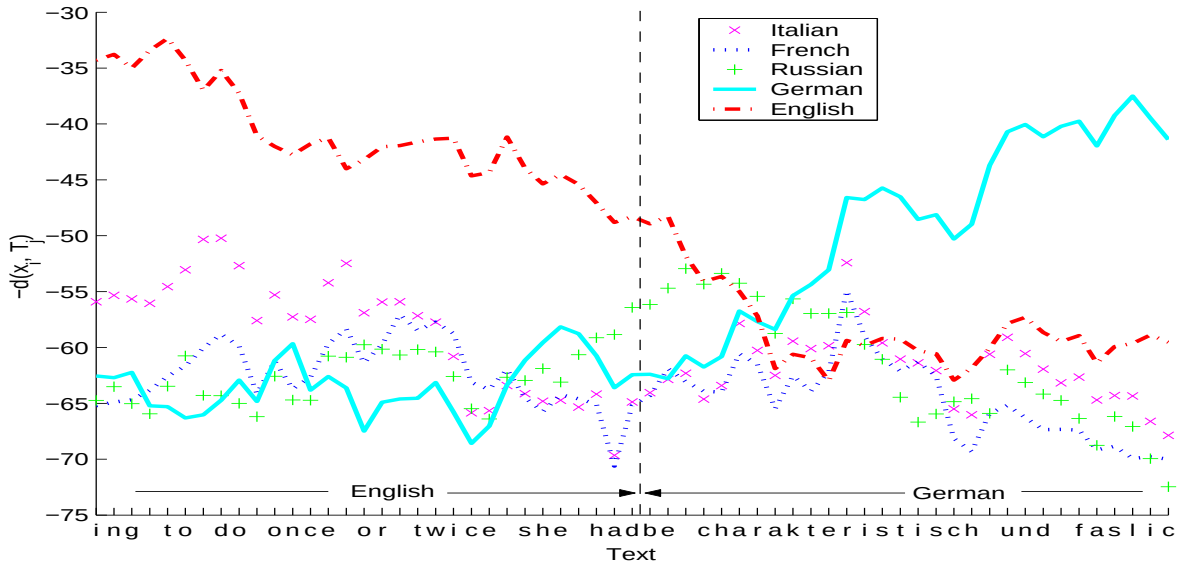


Figure 6.2: **Zoom in on a transition region from English to German**. The text on the x-axis corresponds to $i = 70..130$. Note the correspondence of the transition point into German to the English-like beginning "*be chara*kteristisch..." of the German segment.

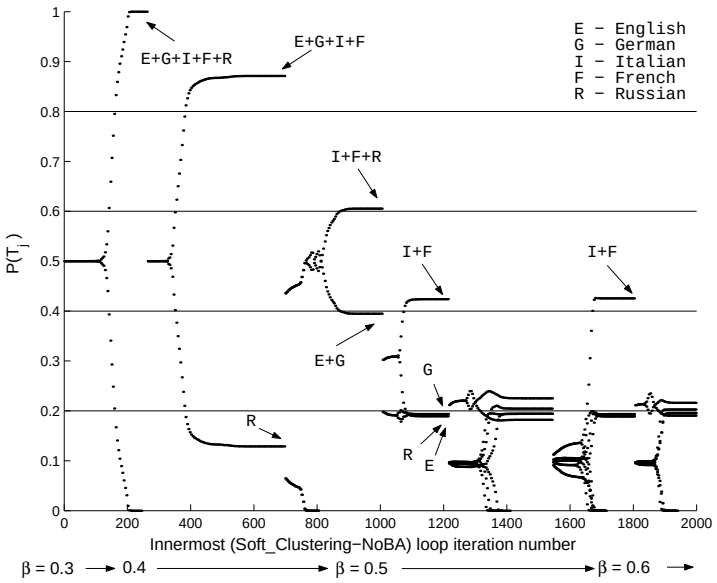


Figure 6.3: **Source separation with Soft_Clustering-NoBA.** We show the probability $P(T_j)$ of each model, as calculated in step 2. of the Soft_Clustering-NoBA procedure (see Fig. 5.7), as a function of cumulative number of this iteration. The values of β at the corresponding places of the clustering evolution process are written below the x axis. Languages, captured by each model after convergence of the clustering process, are designated (models T_j with $P(T_j) \approx 0.2$ capture a single language and therefore are not always labelled). See Sec. 6.1.1 for discussion of this graph.

bifurcations in the graph correspond to $\mathcal{T}$ splitting events (when we split T_j, $P(T_j)$ is equally distributed among the two son models, causing the curve to "fall" by a factor of 2). Curves going down to zero correspond to models loosing all their data; those models are eliminated after convergence. For models incorporating more than one language (and for some "singletons") the languages captured by the model after convergence of Soft_Clustering-NoBA are designated. Because we have 5 languages with equal amount of data from each one, the probability $P(T_j)$ of a model capturing a single source is approximately 0.2.

There are several important observations we may do:

1. After the algorithm separated all the 5 languages ($\mathcal{T}$ grew up to include 5 models) further splits of $\mathcal{T}$ and subsequent execution of Soft_Clustering-NoBA resulted in starvation of the 5 extra models and convergence to the same segmentation we had before the split. This holds true for relatively long range of β (which is increased after each convergence since the number of models does not increase), as may be seen in the zoom out of the graph in Fig. 6.4. Thus we may conclude that the obtained clusters are *stable*.

2. Linguistically similar languages (like English and German; Italian and French) separate at greater values of β (later stages of the segmentation process), suggesting a philological tree structure over the languages. For the 5 languages shown the philological relations

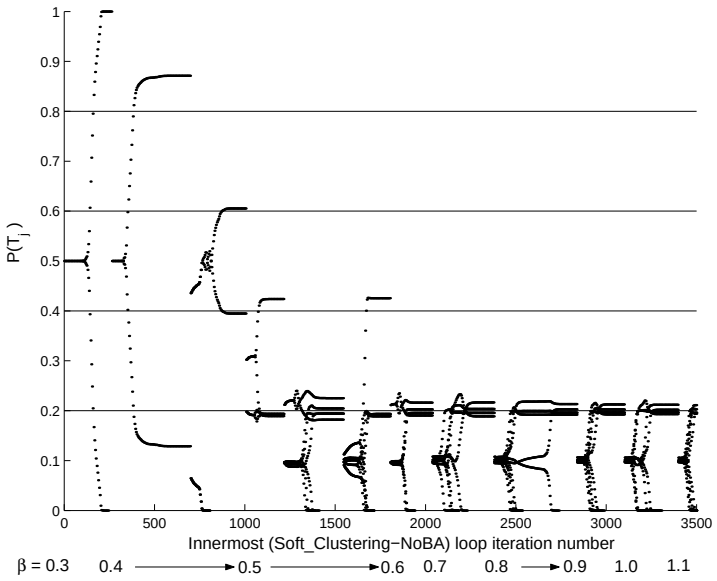


Figure 6.4: **Source separation with Soft_Clustering-NoBA - zoom out.** Zoom out of Fig. 6.3. Observe that the (correct) segmentation obtained after 2000 iterations does not change as we increase β for rather long period, pointing out that the obtained clusters are *stable*.

we got commit to the ones presented in [EYFT98] work, where the tree was built using bottom-up strategy (also based on the statistics of the languages).

3. Put attention on the $\propto$-like shape of curves in some places short after the splits (most clearly seen around the iterations #1250 and #1850). This witnesses that the segmentation process "feels" the underlying data and may correct non-optimal splits.

4. Observe, that at steps 1400-1800 the segmentation converges to 5 models, but after a subsequent split two of them unite back (or, more correctly, the data captured by the two unites back into a single cluster). This happened since 4 models are sufficient to describe the data at the current level of resolution, and the segmentation with 5 models was just some local minima. Thus we see that a split of $\mathcal{T}$ may take the algorithm out of some local minima it occasionally got into.

Running BA till convergence

In Fig. 6.5 we show a graph similar to the one in Fig. 6.3 with the only difference that this time we used the segmentation algorithm based on the BA procedure when executed until convergence. The graph shows $P(T_j)$ as a function of the cumulative number of iteration 2. in the BA procedure (see Fig. 5.1) where $P(T_j)$ is calculated. Comparing Fig. 6.5 with Fig. 6.3 we may observe that:

1. The curves after the bifurcations have a clear $<$-like shape (compared to $\propto$-like in Fig. 6.3). This witnesses that once a "direction" for a separation of models was chosen, it can not

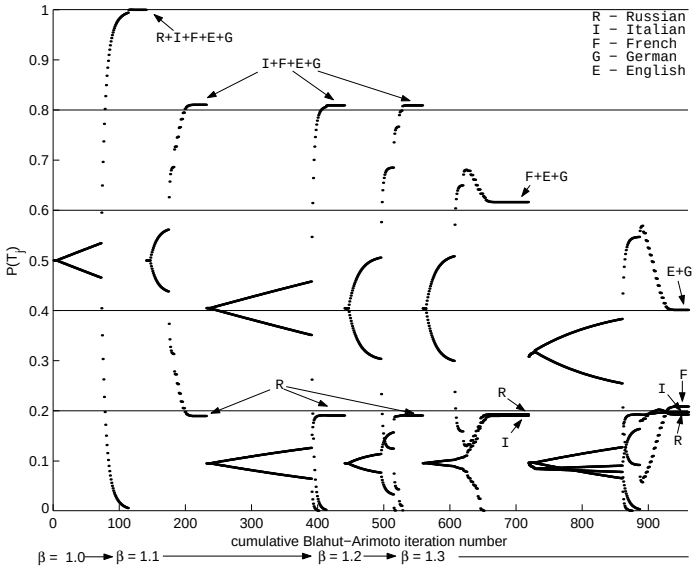


Figure 6.5: **Source separation with BA**. The graph is similar to the one in Fig. 6.3 with the only difference that this time we used segmentation algorithm based on the BA loop when executed until convergence. The graph shows $P(T_j)$ as a function of the cumulative number of the second iteration in the BA procedure, where it is calculated (see Fig. 5.1). Languages captured by each model after convergence of the Soft_Clustering are designated and the values of β corresponding to the iterations are written below the x axis. Compare this graph to the one in Fig. 6.3.

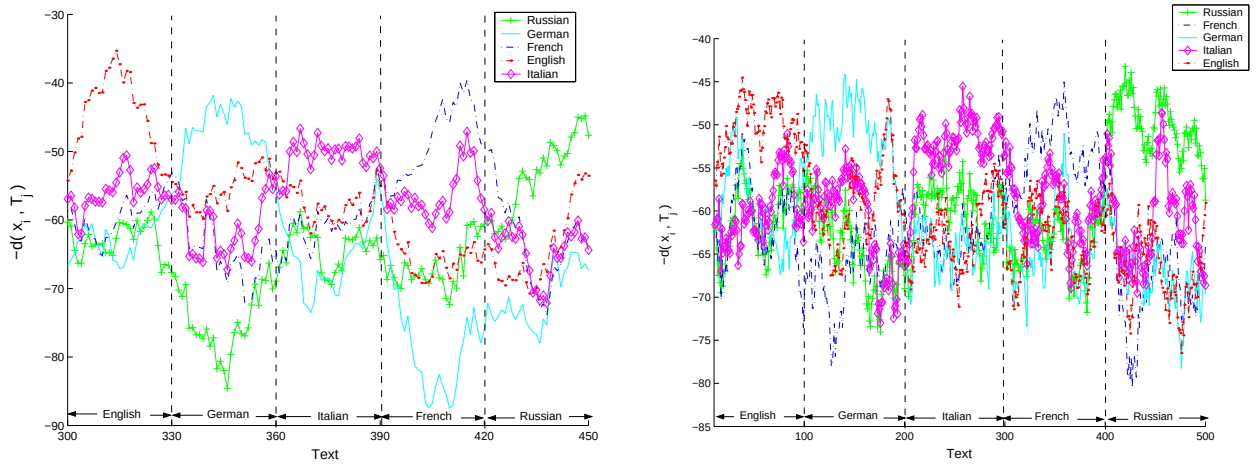
be changed.

2. The languages start separating at β values over 1.0, while in Fig. 6.4 at $\beta = 1.0$ we already had a clear and stable segmentation of the whole text sequence.

We conclude that when BA is executed till the convergence the algorithm is more exposed to local minima since it gets no chance to correct the non-optimal set of models $\mathcal{T}$ it obtained after a random split, and the BA takes it in a random direction to the closest minimum. While for the multilingual data we got the same final separation with both versions of the algorithm (due to relative simplicity of the data and strong attraction of the global optimum), for biological sequences usage of the segmentation algorithm based on Soft_Clustering-NoBA procedure gave significant improvements (actually, we were not able to obtain any results with the BA-based algorithm).

6.1.2 Going toward the limits

In this section we try to find the limits of power of our algorithm. The aim is to build a kind of a benchmark to enable evaluation of the future developments of the algorithm. In all the presented experiments (as well as in the one presented earlier) we use an averaging window of 21 letters size ($M = 10$), as this experimentally appeared to be the best for working with languages.



(a) Very high alternation rate

(b) Very small sample size

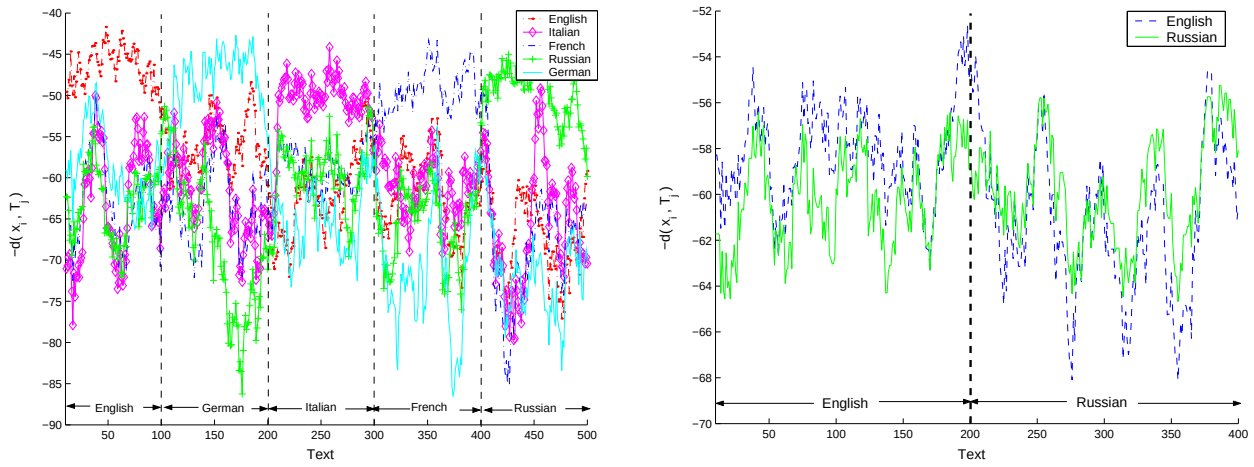
Figure 6.6: **Limits of power.** The graphs are similar to the one in Fig. 6.1: they show $-d(x_i, T_j)$ for selected fragment of text. (a) - the alternation frequency taken is 30 letters (total amount of text is 30000 letters from each language, as in Fig. 6.1). (b) - the alternation rate is 100 letters (as in Fig. 6.1), but the total amount of text from each language is 6000 letters only. One may notice a degradation in the segmentation quality, but the separation is still present.

Working with very high alternation rates

In our first such experiment we try to find the limit of the alternation rate we are able to detect. We keep the amount of data as in the previous example - 30000 letters from each language, and decrease the length of each segment (while increasing their amount). This way we could get down to 30 symbols fragments, i.e. we were switching the languages every 30 letters, which is quite amazing since 30 letters make less than an average sentence. The segmentation we got is presented in Fig. 6.6.a. It should be noted that the segmentation we got was not stable - after subsequent splits the models (mainly the Italian and the French ones) could reunite together and then separate again. But at 40 letters alternation frequency (two times the averaging window size) the segmentation was both clear and stable.

Working with very small amounts of data

In our next experiment we try to find the minimal amount of data needed to perform the segmentation. We kept on alternation frequency of 100 symbols and tried to minimize the number of fragments taken. The low we could get to was 6000 letters from each language (60 fragments), see Fig. 6.6.b for the result. This seems to be an inherent limitation on the amount of data from the side of the PST model we currently work with: the PSTs we got after training them on 6000 letters of text on different languages were 10-15 nodes size (and of depth 1). This seems to be the minimum to make any statistically significant generalizations. We note that this time the segmentation also was not stable at the Italian and French segments. To get a



(a) PST depth limited to 1

(b) PST depth limited to 0

Figure 6.7: **Limiting the depth of PSTs.** (a) - separation of 5 languages with PST depth limited to 1. Compare this graph to the one in Fig. 6.1. (b) - separation of 2 languages with PST depth limited to zero (single root node); the alternation rate between the languages is 200 letters.

stable segmentation there we needed about 8000 letters from each language.

Limiting the depth of PST

In our last experiment we try to limit the depth (and thus power) of the PST model. When limited to 1, it may be seen that the models are still able to distinguish between the languages (see Fig. 6.7.a), but the segmentation quality is more poor, comparing to the one of non-limited PSTs (in Fig. 6.1). The log likelihood (or negative distance between the symbols and corresponding models) also decreases. When the depth is limited to zero (just a single root node), the power of the segmentation algorithm decreases drastically - it can hardly distinguish between two languages when switched only every 200 letters (see Fig. 6.7.b).

Thus we conclude that the power of the PST model plays an important role in the whole algorithm, and further strengthening of the model may award us with even better results.

6.2 Classification of Proteins

In this section we are going to show the potential of applying our algorithm to the problem of protein sequences classification. We start with a very short introduction to the field of molecular biology and current methods for protein sequences analysis. People interested in getting more wide and deep knowledge in those fields are mostly welcome to look in [ABL+94] and [DEKM98]. Then we show some results obtained with our algorithm, mainly from the [BSMT01] work.

6.2.1 Very Short Introduction to Molecular Biology

Proteins are sequences of amino-acids. There are 20 different amino-acids biologists distinguish in nature, thus each protein may be viewed as a piece of text over a 20 letters alphabet. The length of most sequences varies from a few tens to a bit over a thousand amino acids with typical values in the range of few hundreds.

The function of a protein is determined by its sequence. Numerous proteins exhibit a modular architecture, consisting of several sequence domains that often carry specific biological functions (reviewed in [Bor92] and [BK96]). For proteins whose structure has been solved, it can be shown in many cases that the characterized sequence domains are associated with autonomous structural domains. In proteins of various organisms we may find domains that are responsible for similar biochemical functionality. The sequences of those domains will usually be resembling, but not identical. Characterization of a protein family by its distinct sequence domains (also termed 'modules') either directly or through the use of domain 'motifs' or 'signatures' (short sub-segments of the domain that are typical for most members of that family), is crucial for functional annotation and correct classification of newly discovered proteins.

Many methods have been proposed for classification of proteins based on their sequence characteristics. Most of them are based on a seed multiple sequence alignment (MSA - see [DEKM98]) of proteins that are known to be related. The multiple sequence alignment can then be used to characterize the family in various ways: by defining characteristic motifs of the functional sites (as in Prosite, [HBFB99]), by providing a fingerprint that may consist of several motifs (PRINTS-S, [ACF$^+$00]), by describing a multiple alignment of a domain using a hidden Markov model (Pfam, [BBD$^+$00]), or by a position specific scoring matrix (BLOCKS, [HGPH00]). All the above techniques, however, rely strongly on the initial selection of the related protein segments for the MSA, usually hand crafted by experts, and on the quality of the MSA itself. Besides being in general computationally intractable, when remote sequences are included in a group of related proteins, establishment of a good MSA ceases to be an easy task and delineation of the domain boundaries proves even harder. This becomes nearly impossible for heterogeneous groups of proteins, where the shared motifs are not necessarily abundant or do not come in the same order.

The advantage of our algorithm is that it does not attempt any alignment, but rather clusters together regions with similar *statistics*. The regions need not come in the same order, nor they need to be identical - small variations are just a part of the VMM model. In addition, our algorithm is unsupervised - there is no need in prior selection of groups of related proteins, the algorithm will find them even in a bunch of unrelated stuff, as we will show shortly. This is even more attracting since the algorithm may find some new structure or correlations in the data we possibly have not thought about. Thus our approach opens a new promising way to protein sequence analysis, classification and functional annotation.

6.2.2 Experimental results

In this section we demonstrate the results of application of our algorithm to several protein families. We used the modified version of the algorithm, based on the Soft_Clustering-NoBA procedure, that works with sets of multiple strings.

The different training sets were constructed using the Pfam (release 5.4) and Swissprot (release 38 [BA00]) databases. Various sequence domain families were collected from Pfam. In

each Pfam family all members share a domain. An HMM detector is built for that domain based on an MSA of a seed subset of the family domain regions. The HMM is then verified to detect that domain in the remaining family members. Multi-domain proteins therefore belong to to as many Pfam families as there are different characterized domains within them. In order to build realistic, more heterogeneous sets, we collected from Swissprot the *complete sequences* of all chosen Pfam families. Each set now contains a certain domain in all its members, and possibly various other domains appearing anywhere within some members.

There were two types of PST models we got in the process of clustering of the protein data: models that significantly outperform others on relatively short regions (and generally do poor on most other regions) - these we call detectors; and models that perform averagely over all sequence regions - these are "protein noise" (baseline) models. In what is following we analyse what kind of protein segments were selected by the detectors on three exemplary families. In general the "highlighted" segments may be characterized as "segments with highly conserved statistics (sequence), common to at least small amount of the input proteins". Being such, the detected segments may be seen as signatures (or fingerprints) of the domains, though in the cases of very conserved domains the complete domain may be covered by the detector(s). In any case, since living organisms pass through a process of natural selection, we know that only those who have a functioning set of proteins survive. Thus "noisy" segments correspond to less critical sections of proteins and a "mistake" (substitution, insertion or deletion of amino acid) in those sections is possible - therefore we get lots of different variants of those segments. As to segments selected by the detectors - those are vitally important parts of the protein and a mistake there (during the replication process of the corresponding DNA segment) causes loss of functionality of the protein and subsequent death of the organism. Therefore those segments are (almost) the same in all living organisms we see. The amount (or percentage) of proteins sharing a similar segment among all the input proteins may be miserable and the similarity will still be found (in one example we had a domain that was common to only 12 out of 396 input proteins, and it still was altered). This is a clear and strong advantage of our approach compared to MSA, as will be demonstrated here.

In all the following examples we made several independent runs of our algorithm on each chosen family. For each family the different runs converged to the same (stable) final segmentation. In the presented graphs we show the segmentation of *single* representative protein sequences out of the explored families. The Swissprot accession number of the representative sequences shown will be written at the top of each graph.

The Pax Family

Pax proteins (reviewed in [SKG94]) are eukaryotic transcriptional regulators that play critical roles in mammalian development and in oncogenesis. All of them contain a conserved domain of 128 amino acids called the paired or paired box domain (named after the *drosophila* paired gene which is a member of the family). Some contain an additional homeobox domain that succeeds the paired domain. Pfam nomenclature names the paired domain "PAX".

The Pax proteins show a high degree of sequence conservation. One hundred and sixteen family members were used as a training set for the segmentation algorithm. In Fig. 6.8 we superimpose the prediction of all resulting PST detectors over one representative family member. This Pax6 SS protein contains both the paired and homeobox domains. Both have matching

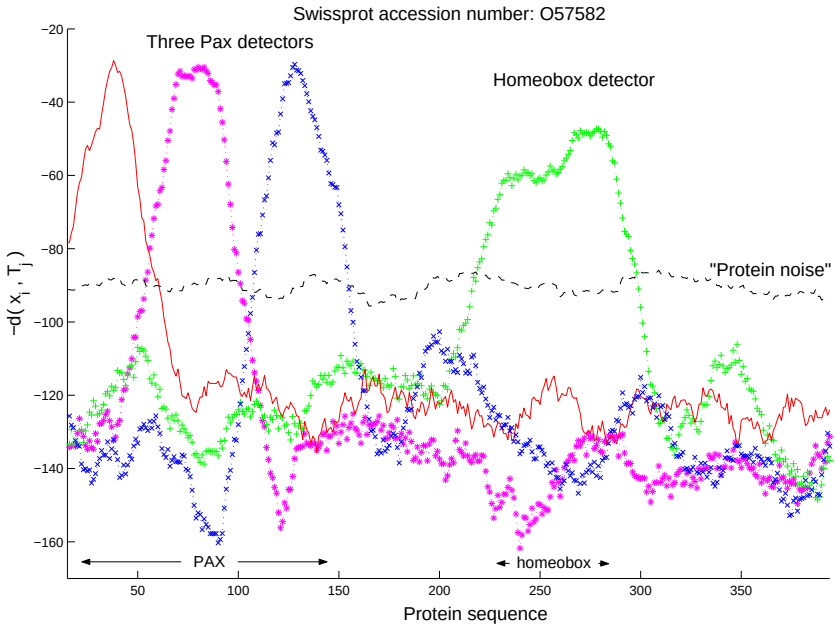


Figure 6.8: **Paired/PAX + homeobox signatures.** The graph shows the segmentation of PAX6 SS protein we got. At the bottom we denote in Pfam nomenclature the location of the two experimentally verified domains. These are in near perfect match here with the high scoring sequence segments.

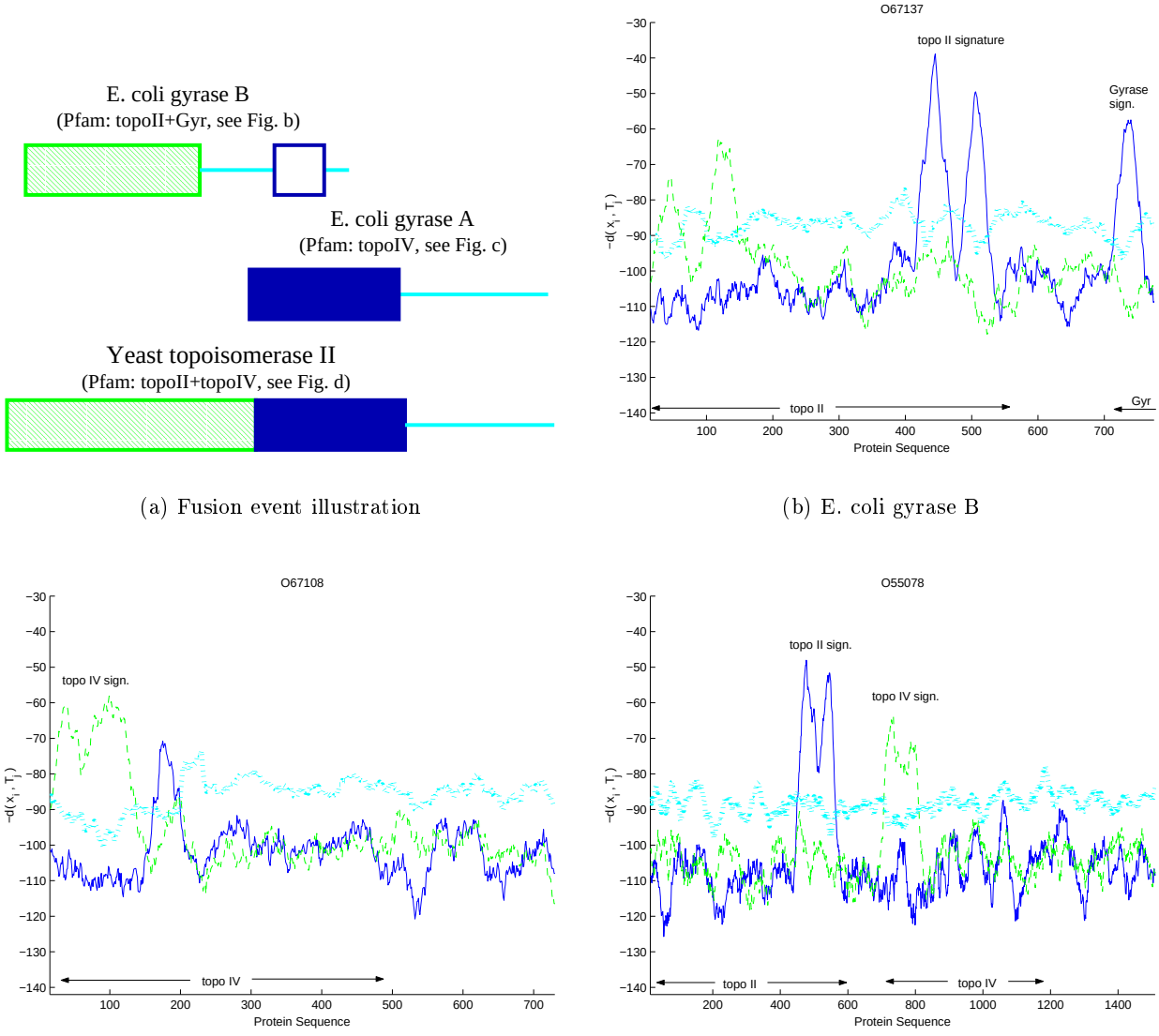
signatures. This also serves as an example where the signatures exactly overlap the domains. The graph of family members not having the homeobox domain contains only the paired domain signature. Note that only about half of the proteins contain the homeobox domain and yet its signature is very clear.

DNA Topoisomerase II

Type II DNA topoisomerases are essential and highly conserved in all living organisms (see [Roc95] for a review). They catalyze the interconversion of topological isomers of DNA and are involved in a number of mechanisms, such as supercoiling and relaxation, knotting and unknotting, and catenation and decatenation. In prokaryotes the enzyme is represented by the *Escherichia coli* gyrase, which is encoded by two genes, gyrase A and gyrase B. The enzyme is a tetramer composed of two gyrA and two gyrB polypeptide chains. In eukaryotes the enzyme acts as a dimer, where in each monomer two distinct domains are observed. The N-terminal domain is similar in sequence to gyrase B and the C-terminal domain is similar in sequence to gyraseA (Fig. 6.9.a). In Pfam 5.4 terminology gyrB and the N-terminal domain belong in the "DNA_topoisoII" family[1], while gyrA and the C-terminal domain belong in the "DNA_topoisoIV" family[2]. Here we term the pairs gyrB/topoII and gyrA/topoIV.

[1] Apparently this family has been sub-divided in Pfam 6 releases.

[2] The name should not be confused with the special type of topoisomerase II found in bacteria, that is also termed topoisomerase IV, and plays a role in chromosome segregation.



(a) Fusion event illustration

(b) E. coli gyrase B

(c) E. coli gyrase A

(d) Yeast topoisomerase II

Figure 6.9: **DNA topoisomerase II.** (a) - Fusion event illustration, adapted from [MPN+99]. The Pfam domain names are added in brackets, together with a reference to our results on a representative homolog. Compare the PST signatures in figures (b)-(d) with the schematic drawing in (a). It is clear that the eukaryotic signature is indeed composed of the two prokaryotic ones, in the correct order, omitting the C-terminus signature of gyrase B (short termed here as "Gyr").

For the analysis we used a group of 164 sequences that included both eukaryotic topoisomerase II sequences and bacterial gyrase A and B sequences (gathered from the union of the DNA_topoisoII and DNA_topoisoIV Pfam 5.4 families). We successfully differentiate them into sub-classes. Fig. 6.9.d describes a representative of the eukaryotic topoisomerase II se-

quences and shows the signatures for both domains, gyrB/topoII and gyrA/topoIV. Fig. 6.9.b and Fig. 6.9.c demonstrate the results for representatives of the bacterial gyrase B and gyrase A proteins, respectively. The *same* two signatures are found in all three sequences, at the appropriate locations. Interestingly, in Fig. 6.9.b in addition to the signature of the gyrB/topoII domain *another* signature appears at the C-terminal region of the sequence. This signature is compatible with a known conserved region at the C-terminus of gyrase B,[3] that is involved in the interaction with the gyrase A molecule.

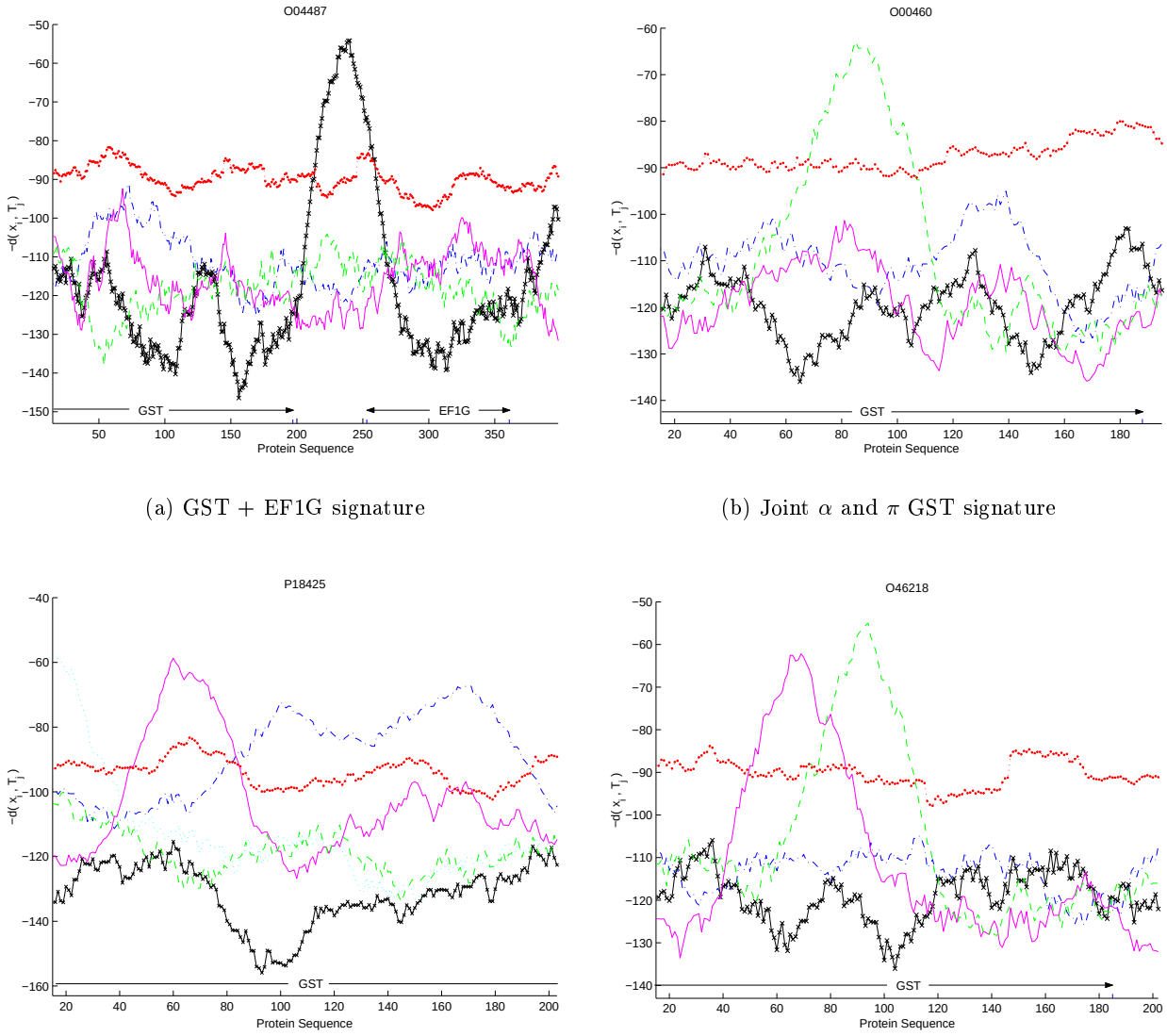
The relationship between the *E. coli* proteins gyrA and gyrB and the yeast topoisomerase II (Fig. 6.9.a) provides a prototypical example of a fusion event of two proteins that form a complex in one organism into one protein that carries a similar function in another organism. Such examples have lead to the idea that identification of those similarities may suggest the relationship between the first two proteins, either by physical interaction or by their involvement in a common pathway [MPN+99, EIKO99]. The computational scheme we present can be useful in search for these relationships.

The Glutathione S-Transferases (GST)

The glutathione S-transferases (GST) represent a major group of detoxification enzymes (reviewed in [HP95]). There is evidence that the level of expression of GST is a crucial factor in determining the sensitivity of cells to a broad spectrum of toxic chemicals. All eukaryotic species possess multiple cytosolic GST isoenzymes, each of which displays distinct binding properties. A large number of cytosolic GST isoenzymes have been purified from rat and human organs. On the basis of their sequences they have been clustered into five separate classes designated class alpha, mu, pi, sigma, and theta GST. The hypothesis that these classes represent separate families of GST is supported by the distinct structure of their genes and their chromosomal location. The class terminology is deliberately global, attempting to include as many GSTs as possible. However, it is possible that there are sub-classes that are specific to a given organism or a group of organisms. In those sub-classes the proteins may share more than 90% sequence identity, but these relationships are masked by their inclusion in the more global class. The classification of a GST protein with weak similarity to one of these classes is sometimes a difficult task. In particular, the definition of the sigma and theta classes is imprecise. Indeed in the PRINTS [ACF+00] database only the three classes, alpha, pi, and mu have been defined by distinct sequence signatures, while in Pfam all GSTs are clustered together, for lack of sequence dissimilarity.

Three hundred and ninety six Pfam family members were segmented jointly by our algorithm, and the results were compared to those of PRINTS (as Pfam classifies all as GSTs). Five distinct signatures were found: (1) A typical weak signature common to many GST proteins that contain no sub-class annotation. (2) A sharp peak after the end of the GST domain appearing exactly in all 12 out of 396 (3%) proteins where the elongation factor 1 gamma (EF1G) domain succeeds the GST domain (Fig. 6.10.a). (3) A clear signature common to almost all PRINTS annotated alpha and most pi GSTs (Fig. 6.10.b). The last two signatures require more knowledge of the GST superfamily. (4) The theta and sigma classes are abundant in nonvertebrates. As more and more of these proteins are identified it is expected that additional classes will be defined. The first evidence for a separate sigma class was obtained by sequence alignments of

[3] Corresponding to the Pfam "DNA_gyraseB_C" family.



(a) GST + EF1G signature

(b) Joint α and π GST signature

(c) S-crystallin GST signature

(d) Putative θ GST signature

Figure 6.10: **Glutathione S-transferases.**

S-crystallins from mollusc lens. Although these refractory proteins in the lens probably do not have a catalytic activity they show a degree of sequence similarity to the GSTs that justifies their inclusion in this family and their classification as a separate class of sigma [BE92]. This class, defined in PRINTS as S-crystallin, was almost entirely identified by the fourth distinct signature (Fig. 6.10.c). (5) Interestingly, the last distinct signature, is composed of two detector models, one from each of the previous two signatures (alpha + pi and S-crystallin) Fig. 6.10.d. Most of these two dozens proteins come from insects, and of these most are annotated to belong to the theta class. Note that many of the GSTs in insects are known to be only very distantly

related to the five mammalian classes. This putative theta sub-class, the previous signatures and the undetected PRINTS mu sub class are all currently further investigated.

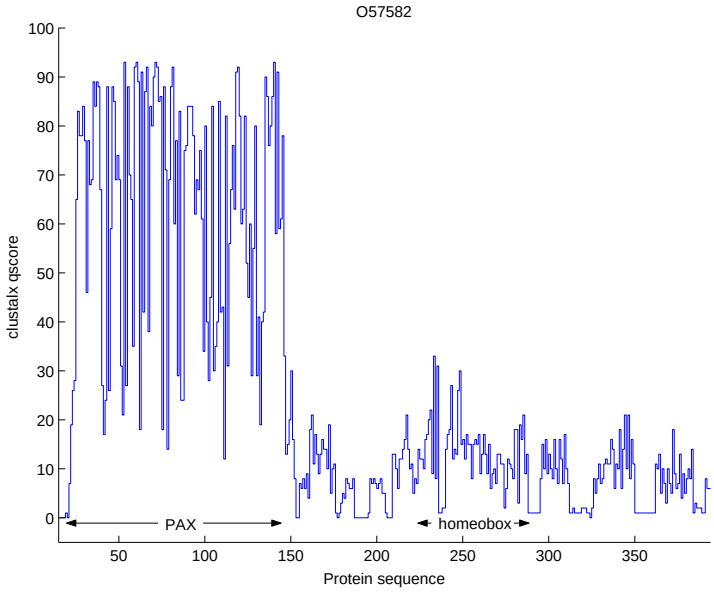


Figure 6.11: **Pax MSA profile conservation.** We plot the clustal X conservation score of the PAX6 SS protein against an MSA of all Pax proteins. While the predominant paired/PAX domain is discerned, the homeobox domain (appearing in about half of the sequences) is lost in the background noise. Compare this graph to the one in Fig. 6.8.

Comparative results

In order to evaluate our findings we have performed three unsupervised alignment driven experiments using the same sets described above: an MSA was computed for each set using clustal X [JTG+98, Linux version 1.81]. We let clustal X compare the level of conservation between individual sequences and the computed MSA profile in each set. Qualitatively these graphs resemble ours, apart from the fact that they do not offer separation into distinct models.

We briefly recount the results we got: the Pax family alignment (shown in Fig. 6.11) discerned the dominant Pax domain, but did not clearly elucidate the homeobox domain existing in about half of the sequences and clearly seen in Fig. 6.8 (compare Fig. 6.11 with Fig. 6.8). For type II topoisomerases the Gyrase B C-terminus unit from Fig. 6.9.b can be discerned from the main unit, but with a much lower peak. And the clear sum of two signatures we obtained for the eukaryotic sequences (Fig. 6.9.d) is lost in noise. In the last and hardest case the MSA approach tells us nothing. All GST domain graphs look nearly identical precluding any possible subdivision. And the 12 (out of 396) instances of the EF1G domain are completely lost at the alignment phase.

Chapter 7

Discussion and Further Work

7.1 Discussion

The sequence segmentation algorithm we describe and evaluate in this work is a combination of several different information theoretic ideas and principles, naturally combined into one new coherent procedure. The core algorithm, the construction of PST, is essentially a source coding *loss-less compression* method. It approximates a complex stochastic sequence by a Markov model with variable memory length. The power of this procedure, as demonstrated on both natural texts and on protein sequences [RST96, BY01], 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. In addition, the new algorithm is completely non-parametric and thus perfectly suits for unsupervised learning problems.

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. We treat the PST as a model of a single statistical source and use the rate distortion framework 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 [FST98]) 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 and the total amount of data from each source, our segmentation scheme should perform well.

In our experiments with segmentation of multilingual text sequences (mixtures of European languages) we demonstrated the ability of our algorithm to differentiate between the languages with a precision of few letters, even when the languages are switched every 30-40 letters. The

minimal amount of text (from each language) needed to perform any segmentation appeared to be around 6000-8000 letters.

Our experiments with protein families demonstrated a number of clear advantages of the proposed algorithm: it is fully automated; it does not require or attempt an MSA of the input sequences; it handles heterogeneous groups well and locates domains appearing only few times in the data; by nature it is not confused by different module orderings within the input sequences; it appears to seldom generate false positives; and it is shown to surpass HMM clustering in at least one hard instance.

In our opinion the new tool may suggest a new perspective on protein sequence organization at large. Statistical conservation is unlike conventional sequence conservation. Regions may be statistically identical, while completely different from the alignment point of view (like in the case of multilingual texts). We hope that this new, much more flexible notion of sequence conservation will eventually help better understand the constraints shaping the world of known proteins.

7.2 Further Work

There is a plenty of directions to take our algorithm to both in the applicative and in the theoretical fields.

In the applicative field it would be extremely interesting to run our algorithm on all known proteins. The top-down organization of proteins may bring new interesting insights into the complicated world of biology. We also think about trying our algorithm on additional types of datasets, such as DNA sequences, network flow, spike trains, speech signals, stock rates, etc.

In the theoretical aspect we see two independent parts in our algorithm: training of new set of models given a segmentation, and finding a "good"[1] segmentation given a set of models.

We think that there is still place for improvement of the PST model. We may try to improve the compression ratio by uniting together son nodes with similar statistics (like it is done in the [RST95] work). We may also try to improve the time complexity of the algorithm by embedding the ideas of the [AB00] work to our case of MDL-oriented learning of PSTs. The MDL coding of a single node in PST may also be improved, as was discussed in our talk with Adi Wynner.

As to the segmentation of a sequence with a given set of models, our main aim at the moment is to get rid of the fixed size averaging window. We think that the way to do this lies through segmentation process similar to as it is done in HMMs. Hopefully, with a new segmentation procedure we will also be able to prove the convergence of our algorithm. It seems like we should obtain and prove a monotonic decrease of the total code length, and not just an increase of the likelihood of the data, as it is done in the proof of convergence of the EM algorithm. And it seems like HMM-like formulation of the segmentation procedure may help us with this.

We denote by $h_i = x_1..x_{i-1}$ the "history" preceding x_i. Then $P(T_j|h_i)$ is the prior probability that T_j is the generating model at index i of the sequence, and $P(T_j|x_i, h_i)$ is the posterior of T_j. This time we define the distance between x_i and T_j to be negative log likelihood T_j gives to x_i *only* - we do not use the averaging window:

$$d(x_i, T_j) = -\ln P(x_i|T_j)$$

[1] Our experience with the BA points out that possibly we do not always want to get the best segmentation right away.

Thus our assignment probabilities in 1. of the Soft_Clustering-NoBA procedure (Fig. 5.7) are defined as:

$$P(T_j|x_i, h_i) = \frac{P(T_j|h_i)e^{\beta \ln P(x_i|T_j)}}{Z(x_i, \beta)}$$

Where $Z(x_i, \beta)$ is a normalization factor.

The sequential dependencies in the data (the limitation of the switching rate) are now expressed through the prior probabilities that are not constant over the sequence any more:

$$P(T_j|h_i) = \sum_{\alpha} P(T_\alpha|x_{i-1}, h_{i-1})P(T_\alpha \to T_j)$$

Here $P(T_\alpha \to T_j)$ is the probability that a model T_α is switched to a model T_j at place i, and $P(T_\alpha|x_{i-1}, h_{i-1})$ is the posterior probability of T_α at index $i-1$.

The only thing left to define now are the transition probabilities $P(T_\alpha \to T_\beta)$. If we define them to be some arbitrary constant: $P(T_\alpha \to T_\beta) = \lambda_{\alpha\beta}$, we will get averaging windows of exponential form (which, we think, should be better than uniform averaging windows we have now). Another option is to try to estimate those probabilities, as it is done in the Baum-Welsh algorithm (see [Rab86] for an overview). One more option is to calculate $\lambda_{\alpha\beta}$ using MDL principles and to prove some upper bound on the compression ratio we get compared to the optimal one we could get. I.e. we may try to improve the results of [HW98] for the case when the predicting models (experts) are known.

Bibliography

[AB00] Alberto Apostolico and Gill Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *J. Comput. Biol.*, 7(3):381–393, 2000.

[ABL+94] Bruce Alberts, Dennis Bray, Julian Lewis, Martin Raff, Keith Roberts, and James D. Watson. *Molecular Biology of the Cell.* Garland Publishing, 3 edition, 1994.

[ACF+00] T.K. Attwood, M.D. Croning, D.R. Flower, A.P. Lewis, J.E. Mabey, P. Scordis, J.N. Selley, and W. Wright. PRINTS-S: the database formerly known as PRINTS. *Neuc. Acids Res.*, 28(1):225–227, 2000.

[Ari72] S. Arimoto. An algorithm for computing the capacity of discrete memoryless channel. 18:14–20, 1972.

[BA00] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Neuc. Acids Res.*, 28(1):45–48, 2000.

[BBD+00] A. Bateman, E. Birney, R. Durbin, S.R. Eddy, K.L. Howe, and E.L. Sonnhammer. The Pfam protein families database. *Neuc. Acids Res.*, 28(1):263–266, 2000.

[BE92] T.M. Buetler and D.L. Eaton. Glutathione S-transferases: amino acid sequence comparison, classification and phylogentic relationship. *Environ. Carcinogen. Ecotoxicol. Rev.*, C10:181–203, 1992.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon press, Oxford, 1995.

[BK96] P. Bork and E.V. Koonin. Protein sequence motifs. *Curr. Opin. Struct. Biol.*, 6(3):366–376, 1996.

[Bla72] R. Blahut. Computation of channel capacity and rate distortion functions. 18:460–473, 1972.

[Bor92] P. Bork. Mobile modules and motifs. *Curr. Opin. Struct. Biol.*, 2:413–421, 1992.

[BRY98] A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Trans. Info. Theory*, 44:2743–2760, 1998.

[BSMT01] Gill Bejerano, Yevgeny Seldin, Hanah Margalit, and Naftali Tishby. Markovian domain fingerprinting: statistical segmentation of protein sequences. *Bioinformatics*, 17(10):927–934, 2001.

[BY01] Gill Bejerano and Golan Yona. Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinform.*, 17(1):23–43, 2001.

[Can33] F. Cantelli. Sulla determinazione empirica della leggi di probabilita. *G. Inst. Ital. Attuari*, 4, 1933.

[Cha66] G.J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the Association of Computing Machinery*, 13:547–569, 1966.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[Csi74] Imre Csiszar. On the computation of rate distortion functions. *IEEE Trans. Info. Theory*, 20:122–124, 1974.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. John Wiley & Sons, New York, NY, 1991.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[DGL96] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer, 1996.

[DH73] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.

[DHS01] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* John Wiley & Sons, New York, 2001.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, 39(1):1–38, 1977.

[EIKO99] A.J. Enright, I. Iliopoulos, N.C. Kyrpides, and C.A. Ouzounis. Protein interaction maps for complete genomes based on gene fusion events. *Nature*, 402(6757):86–90, 1999.

[EYFT98] Ran El-Yaniv, Shai Fine, and Naftali Tishby. Agnostic classification of markovian sequences. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Adv. Neural Inform. Proc. Sys. (NIPS) 10*, volume 10, pages 465–471. The MIT Press, 1998.

[Fel71] Feller. *An Introduction to Probability Theory and its Applications.* John Wiley & Sons, New York, 1971.

[Fis22] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London*, Series A 222:309–368, 1922.

[Fis25] R. A. Fisher. Theory of statistical estimation. *Trans. Cambridge Philos. Soc.*, 22:700, 1925.

[FR95] Yoav Freund and Dana Ron. Learning to model sequences generated by switching distributions. In *Comput. Learn. Theory (COLT) 8*, volume 8, pages 41–50, New York, NY, July 1995. ACM press.

[FST98] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Mach. Learn.*, 32:41–62, 1998.

[Gli33] V. Glivenko. Sulla determinazione empirica di probabilita. *G. Inst. Ital. Attuari*, 4, 1933.

[HBFB99] K. Hofmann, P. Bucher, L. Falquet, and A. Bairoch. The PROSITE database, its status in 1999. *Neuc. Acids Res.*, 27(1):215–219, 1999.

[HGPH00] J. G. Henikoff, E. A. Greene, S. Pietrokovski, and S. Henikoff. Increased coverage of protein families with the Blocks database servers. *Neuc. Acids Res.*, 28(1):228–230, 2000.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[Hof97] Thomas Hofmann. *DATA CLUSTERING AND BEYOND: A Deterministic Annealing Framework for Exploratory Data Analysis*. Shaker Verlag, 1997.

[HP95] J.D. Hayes and D.J. Pulford. The glutathione S-transferase supergene family: regulation of GST and the contribution of the isoenzymes to cancer chemoprotection and drug resistance. *Cri. Rev. Biochem. Mol. Biol.*, 30(6):445–600, 1995.

[HW98] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151, 1998.

[JTG+98] F. Jeanmougin, J. Thompson, M. Gouy, D. Higgins, and T. Gibson. Multiple sequence alignment with clustal X. *Trends Biochem. Sci.*, 23:403–405, 1998.

[KL51] S. Kullback and R. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22:79–86, 1951.

[Kol33] A.N. Kolmogorov. Sulla determinazione empirica di una leggi di distribuzione. *G. Inst. Ital. Attuari*, 4, 1933.

[Kol65] A.N. Kolmogorov. Three approaches to the quantitative denition of information. *Problems of Information and Transmission*, 1:1–7, 1965.

[KT81] R.E. Krichevsky and V.K. Trofimov. The performance of universal coding. *IEEE Trans. Info. Theory*, IT-27:199–207, 1981.

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, Massachusetts, 1994.

[MK97] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. 1997.

[MPN+99] E.M. Marcotte, M. Pellegrini, H.L. Ng, D.W. Rice, T.O. Yeates, and D. Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–753, 1999.

[Rab86] L. R. Rabiner. Tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1986.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Roc95] J. Roca. The mechanisms of DNA topoisomerases. *Trends in Biol. Chem.*, 20:156–160, 1995.

[Ros98] Kenneth Rose. Deterministic annealing for clustering, compression, classification, regression and related optimization problems. *IEEE Trans. Info. Theory*, 80:2210–2239, November 1998.

[RST95] Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proc. 8th Annu. Conf. on Comput. Learning Theory*, pages 31–40. ACM Press, New York, NY, 1995.

[RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Mach. Learn.*, 25:117–149, 1996.

[SBT01a] Yevgeny Seldin, Gill Bejerano, and Naftali Tishby. Unsupervised segmentation and classification of mixtures of Markovian sources. In *Proceedings of the* 33^{rd} *Symposium on the Interface of Computing Science and Statistics*, 2001.

[SBT01b] Yevgeny Seldin, Gill Bejerano, and Naftali Tishby. Unsupervised sequence segmentation by a mixture of switching variable memory Markov sources. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 513–520, 2001.

[Sha48] C.E. Shanon. A mathematical theory of communication. *Bell Sys. Tech. Journal*, 27:379–423, 623–656, 1948.

[SKG94] E. T. Stuart, C. Kioussi, and P. Gruss. Mammalian Pax genes. *Annu. Rev. Genet.*, 28:219–236, 1994.

[Sol60] R.J. Solomonoff. A preliminary report on a general theory of inductive inference. 1960.

[TPB99] Naftali Tishby, Fernando Pereira, and William Bialek. The information bottleneck method. In *Allerton Conference on Communication, Control and Computation*, volume 37, pages 368–379. 1999.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications, and Control. John Wiley & Sons, New York, NY, 1998.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[VC81] V. Vapnik and A. Chervonenkis. Necessary and sufficient conditions for the uniform convergence of means to their expectations. *Theory of Probability and its Applications*, 26(3):532–553, 1981.

[WB68] C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–195, 1968.

[Wil98] F. M. J. Willems. The context-tree weighting method: extensions. *IEEE Trans. Info. Theory*, pages 792–798, March 1998.

[WST95] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Trans. Info. Theory*, 41(3):653–664, May 1995.

[Yon99] Golan Yona. *Methods for Global Organization of all Known Protein Sequences*. PhD thesis, The Hebrew University of Jerusalem, 1999.

[YS68] S. J. Yakowitz and J. D. Spragins. On the identifiability of finite mixtures. *Annals of Mathematics and Statistics*, 39:209–214, 1968.