12

Kernels from generative models

It is often the case that we know something about the process generating the data. For example, DNA sequences have been generated through evolution in a series of modifications from ancestor sequences, text can be viewed as being generated by a source of words perhaps reflecting the topic of the document, a time series may have been generated by a dynamical system of a certain type, 2-dimensional images by projections of a 3-dimensional scene, and so on.

For all of these data sources we have some, albeit imperfect, knowledge about the source generating the data, and hence of the type of invariances, features and similarities (in a word, patterns) that we can expect it to contain. Even simple and approximate models of the data can be used to create kernels that take account of the insight thus afforded.

Models of data can be either deterministic or probabilistic and there are also several different ways of turning them into kernels. In fact some of the kernels we have already encountered can be regarded as derived from generative data models.

However in this chapter we put the emphasis on generative models of the data that are frequently pre-existing. We aim to show how these models can be exploited to provide features for an embedding function for which the corresponding kernel can be efficiently evaluated.

Although the emphasis will be mainly on two classes of kernels induced by probabilistic models, P-kernels and Fisher kernels, other methods exist to incorporate generative information into kernel design. Indeed in some sense every kernel can be regarded as embodying our generative assumptions about the data.

12.1 P-kernels

A very general type of kernel can be obtained by defining a joint probability distribution $P(x, z)$ on pairs of data items (x, z) from the product space $X \times X$ of a finite or countably infinite input space X. We assume the joint probability $P(x, z)$ does not depend on the order of the items so that P is symmetric, but to be a kernel

$$\kappa(x, z) = P(x, z),$$

it must also satisfy the finitely positive semi-definite property.

Definition 12.1 [P-kernel] A probability distribution $P(x, z)$ over a product space $X \times X$ that satisfies the finitely positive semi-definite property will be known as a P-kernel on the set X. ■

Given a kernel κ on a countably infinite space X, we can create a P-kernel that is essentially identical to κ provided

$$\sum_{x \in X} \sum_{z \in X} \kappa(x, z) = M < \infty,$$

since the distribution

$$P(x, z) = \frac{1}{M} \kappa(x, z)$$

satisfies the definition of a P-kernel.

On the other hand, not all distributions will be P-kernels, as the simple example of a two element set

$$X = \{x_1, x_2\},$$

with probabilities

$$\begin{aligned} P(x_1, x_1) &= P(x_2, x_2) = 0; \\ P(x_1, x_2) &= P(x_2, x_1) = 0.5 \end{aligned}$$

shows, since the matrix

$$\begin{pmatrix} 0 & 0.5 \\ 0.5 & 0 \end{pmatrix}$$

indexed by x_1, x_2 has eigenvalues $\lambda = \pm 0.5$.

We have presented P-kernels for countably infinite input spaces since these are the types of data for which we take this approach. For uncountably infinite sets a P-kernel would be defined by a probability density function $p(x, z)$ again assumed symmetric and required to satisfy the finitely positive semi-definite property. A kernel κ would be equivalent to such a P-kernel through a renormalisation provided

$$\int_X \int_X \kappa(x,z)\, dxdz = M < \infty.$$

It follows for example that any bounded kernel over a compact domain can be regarded as a P-kernel.

Hence, a P-kernel is a very broad class of kernels that by itself does not provide interesting insights. Its real interest arises when we study how one might generate such joint probability distributions.

As a simple example consider a probability distribution $P(x)$ on the finite or countably infinite set X. We can define a 1-dimensional P-kernel using the following product

$$\kappa(x,z) = P(x)P(z).$$

This follows immediately from the observation that this kernel corresponds to the 1-dimensional embedding

$$\phi : x \longmapsto P(x).$$

This example suggests the following definition for a probabilistic model.

Definition 12.2 [Data model] A model m of a data generation process is an artificial distribution that assigns to each data item x a probability $P(x|m)$ of being generated. This can be seen as an estimate of the true distribution that generated the data. In practice we do not know precisely how the data was generated, but we may have a class of possible models M and a prior distribution P_M or belief as to which model was more likely to have generated the data. This is encoded in a distribution $P_M(m)$ over the set of models, where m is used to denote the individual model possibly described by a set of parameters. Hence, for example different settings of hidden variables are viewed as different models. ■

Remark 12.3 [Modelling process] The definition of a data model only requires that it be a probability distribution. In practice we typically define the distribution in a way that reflects our understanding of the processes that actually generate the real data. It is for this reason that the term model is used since the distribution results from a model of the generation process.

■

The first part of this chapter considers probability distributions arising from data models that give rise to more general and interesting classes of P-kernels.

12.1.1 Conditional-independence (CI) and marginalisation

Building on the example at the end of the previous section, we can create a joint distribution that is positive semi-definite by combining a number of different 1-dimensional distributions $P(x, z|m) = P(x|m)P(z|m)$. We choose m at random from a set M to combine the different distributions as follows

$$P(x, z) = \sum_{m \in M} P(x, z|m)P_M(m) = \sum_{m \in M} P(x|m)P(z|m)P_M(m),$$

where we have used $P_M(m)$ to denote the probability of drawing the distribution parametrised by m. We could again consider a probability density function in the case that M were an uncountably infinite set.

If we view the set M as a set of potential models of the data distribution each weighted according to our prior belief $P_M(m)$ in that particular model, the distribution corresponds to our prior estimation of the probability that x and z arise as a pair of independently generated examples. It is this view that also suggests the idea of marginalisation. If we view $P(x, z|m) P_M(m)$ as a probability distribution over triples (x, z, m), then the probability $P(x, z)$ is its marginal distribution, since it involves integrating over the unknown variable m. For this reason the kernels are sometimes referred to as marginal kernels.

In general, marginalising does not guarantee that the distribution satisfies the finitely positive semi-definite property. In our case it follows from the combination of marginalising over distributions each of which is finitely positive semi-definite because of the independence assumption in the generation of x and z. For this reason we say that x and z are conditionally independent given m. Hence, the assumption of conditional independence

$$P(x, z|m) = P(x|m)P(z|m)$$

is a sufficient (but not necessary) condition for positive semi-definiteness.

This class of kernels, obtained by marginalisation of conditionally independent functions, is extremely general, and can be efficiently computed for many classes of functions, using algorithmic tools developed in the theory of probabilistic graphical models. This often allows us to turn a generative model of the data into a kernel function. The dimensions of the embedding space are indexed by the elements of a set of probabilistic models $\phi_m(x) = P(x|m)$ as indicated in the following definition summarising the approach.

Definition 12.4 [Marginalisation kernels] Given a set of data models M and a prior distribution P_M on M, we can compute the probability that a pair of examples x and z is generated together

$$P_M(x,z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m).$$

If we consider the projection function

$$\boldsymbol{\phi} : x \longmapsto (P(x|m))_{m \in M} \in F,$$

in a feature space F indexed by M, $P_M(x,z)$ corresponds to the inner product

$$\langle f, g \rangle = \sum_{m \in M} f_m g_m P_M(m)$$

between $\boldsymbol{\phi}(x)$ and $\boldsymbol{\phi}(z)$. It follows that $P_M(x,z)$ is a P-kernel. We refer to $P_M(x,z)$ as the *marginalisation kernel* for the model class M. ■

12.1.2 Representing multivariate distributions

Defining a data model requires us to create a class of multivariate probability distributions. The simplest form for such distributions is to assume independence between the different coordinates and hence use a collection of independent 1-dimensional distributions. Once we allow interactions between the different variables, the number of degrees of freedom increases substantially. For example to specify a multivariate Gaussian distribution we need $O(n^2)$ parameters to give the covariance matrix.

In view of these complications it is natural to search for models that lie somewhere between complete independence of the variables and full dependence. Graphical models are a representation that allows us to specify a dependence structure in the form of a directed acyclic graph whose nodes are the variables. The distribution of the variable A that labels a node becomes independent if we specify the values of the set $\mathcal{N}^-(A)$ of variables on vertices with edges $(B \to A)$ pointing at A. Hence, if the labelling $A_1, \ldots, A_n$ of the nodes is such that there are no edges from later to earlier vertices, then the distribution can be computed using

$$P(A_1, \ldots, A_n) = \prod_{i=1}^{n} P\left(A_i | \mathcal{N}^-(A_i)\right).$$

Specifying the individual conditional probabilities might involve a lookup table in the case of binary variables or some other distribution for more complex variables. Note that one can interpret an edge from node A to node B as implying that A is a partial cause for B.

Hence graphical models provide a compact representation of a joint probability distribution. For example the joint distribution

$$P(A, B, C, D) = P(A)P(B|A)P(C|A)P(D|C, B)$$

can be represented by the graphical model in Figure 12.1.

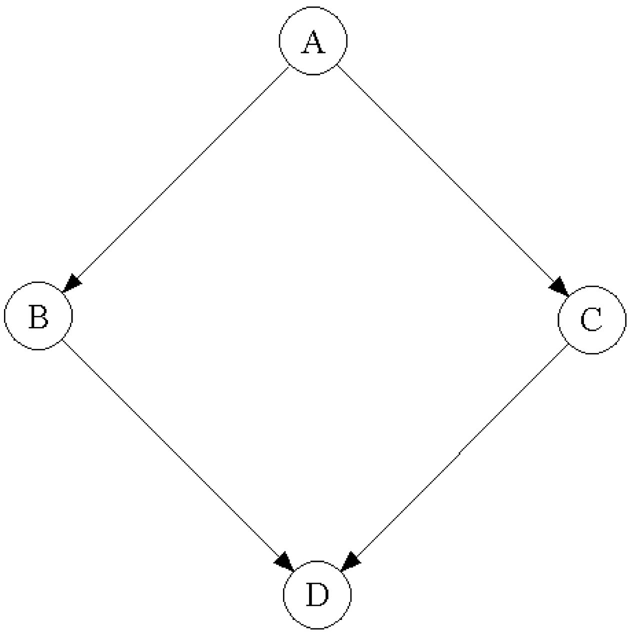

Fig. 12.1. A simple example of a graphical model.

So far we have considered graphical models in which the nodes are in one-to-one correspondence with the observed variables. In practice we may not be able to observe all of the relevant variables. In this case we distinguish between the observables and the so-called hidden variables, with the overall graphical model being defined over both sets of variables.

An example of a graphical model with hidden states is shown in Figure

12.3. It shows a sequence of hidden variables $(h_1, \ldots, h_n)$ linked so that the next variable is conditionally dependent on the previous hidden variable. Each hidden variable h_i conditions a corresponding distribution for the observable variable s_i. This is known as a 1-stage Markov model. We can also represent this by a sequence of nodes linked into a path, where the ith node has a state variable h_i that can take a number of possible values and the node emits a symbol depending on the value of h_i, while the value of h_i depends probabilistically on h_{i-1}. Hence, the symbol emission has been included into the function of the node.

We can make the diagram more explicit by creating a node for each state, that is each state variable–value pair. In this view the transitions take us from one state to another and we have a probabilistic finite state automata whose nodes emit symbols. We prefer to take this general framework as our definition of a hidden Markov model.

Definition 12.5 [Hidden Markov model] A hidden Markov model (HMM) M comprises a finite set of states A with an initial state a_I, a final state a_F, a probabilistic transition function $P_M(a|b)$ giving the probability of moving to state a given that the current state is b, and a probability distribution $P(\sigma|a)$ over symbols $\sigma \in \Sigma \cup \{\varepsilon\}$ for each state $a \in A$. ■

A hidden Markov model generates strings probabilistically by initialising the state to a_I and repeatedly performing random transitions each followed by a random generation of a symbol, until the state a_F is reached.

12.1.3 Fixed length strings generated by a hidden binomial model

The kernel presented in this subsection is a somewhat 'degenerate' example of an important class of models. Nonetheless its discussion sets the scene for two more complex generative models also for fixed length symbol sequences, each based on different assumptions about the hidden generating mechanism. These will be presented in the following two subsections followed by extensions to handle the case of strings of different lengths.

Consider comparing symbol strings that have the same length n with individual symbols drawn from a finite alphabet Σ. We assume that the symbols are generated by 'emission' from a sequence of hidden state variables represented by a string of the same length $h = (h_1 \ldots h_n)$, where the individual emission probabilities are independent that is

$$P(s|h) = \prod_{i=1}^{n} P(s_i|h_i),$$

and as posited in the general framework the probabilities of generating two strings s and t are also treated as independent. This generation model is illustrated in Figure 12.2. Note that to cast this in the framework of

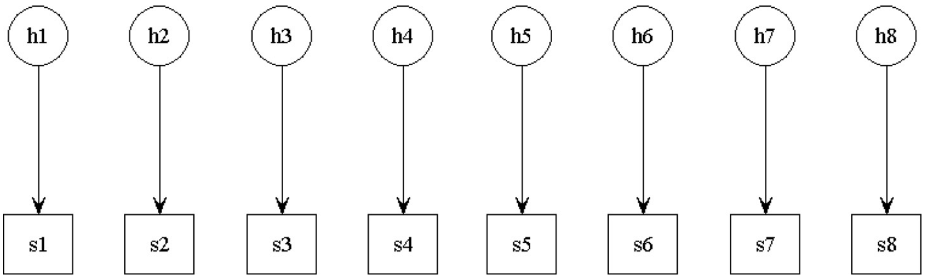

Fig. 12.2. A simple hidden model for generating fixed length strings.

Definition 12.5, we would need to expand each state variable into the set of states that it can assume and include transitions from the set of states for h_i to those for h_{i+1}, $i = 1, \ldots, n-1$, as well as including transitions from the initial state a_I and to the final state a_F. An example of this type of (formal) construction is given in the next subsection.

For the time being we leave the form of the distribution $P(s_i|h_i)$ unspecified, though there is a further assumption that the probabilities of the hidden states $P(h)$ can also be decomposed into a product over the individual symbols

$$P(h) = \prod_{i=1}^{n} P(h_i)$$

so that the kernel involves summing over all hidden strings and marginalising the probability to obtain the marginalisation kernel

$$\begin{aligned} \kappa(s,t) &= \sum_{h\in\Sigma^n} P(s|h)P(t|h)P(h) \\ &= \sum_{h\in\Sigma^n} \prod_{i=1}^{n} P(s_i|h_i)P(t_i|h_i)P(h_i). \end{aligned}$$

We have already encountered sums of products within kernel calculations when we considered the kernels defined by summing over all paths in a graph. This form will also prove a recurring theme in this chapter. The sum is over a large number of hidden models, each of which can be used to compute a probability by means of a product. The key idea is that in certain important cases this can be computed as the product of sums by effectively exchanging the two operators, and hence achieving a significant computational saving. This will motivate a number of dynamic programming methods, but in the specific case considered here we will be able to obtain a closed form solution.

We can write

$$\begin{aligned}\kappa(s,t) &= \sum_{h\in\Sigma^n}\prod_{i=1}^{n} P(s_i|h_i)P(t_i|h_i)P(h_i) \\ &= \prod_{i=1}^{n}\sum_{\sigma\in\Sigma} P(s_i|\sigma)P(t_i|\sigma)P(\sigma), \end{aligned} \tag{12.1}$$

where the expression on the right-hand side of the first line can be obtained by applying the distributive law to the product of sums given in the second line. The variable h_i tells us which term to take in the ith factor of the product with the overall expression given by the sum over all possible choices, that is over all strings h.

We have presented this example more as an illustration of the swapping step. In this case the result is rather trivial, since we could pre-compute a table of

$$P(a,b) = \sum_{\sigma\in\Sigma} P(a|\sigma)P(b|\sigma)P(\sigma), \text{ for } a,b\in\Sigma,$$

and then the kernel could be evaluated as

$$\kappa(s,t) = \prod_{i=1}^{n} P(s_i,t_i).$$

This is a simple product of base kernels

$$\kappa_i(s,t) = P(s_i,t_i)$$

and so could also be viewed as an ANOVA kernel of degree n.

It does, however, also illustrate a general property of P-kernels, namely that if we create P-kernels for each part X_i of a set of n disjoint components of data items from a set $X = X_1\times\cdots\times X_n$, their product also becomes a P-kernel, since

$$\sum_{s\in X}\sum_{t\in X} P(s,t) = \sum_{s\in X}\sum_{t\in X}\prod_{i=1}^{n} P(s_i,t_i)$$

$$= \prod_{i=1}^{n} \sum_{s_i \in X_i} \sum_{t_i \in X_i} P(s_i, t_i)$$
$$= \prod_{i=1}^{n} 1 = 1.$$

Remark 12.6 [States and symbols] It is worth noting that although we discussed this example using the same set of symbols for the strings as for the states, these could be defined over different sets. In view of our final observations this will not have a very significant impact in this example, but we will see that in the next subsection this does afford us extra flexibility. ■

12.1.4 Fixed length strings generated by a hidden markov model

The model above was not very realistic, but did demonstrate the type of computations that we are considering in this section, particularly the idea of marginalising over sums of products. For our next model we continue with the assumptions that the two strings s and t have fixed length n and are composed of symbols from an alphabet Σ. Furthermore we assume that they have been generated by a hidden model M, whose elements are represented by strings h of n states each from a set A, and that each symbol is generated independently, so that

$$P(s,t|h) = \prod_{i=1}^{n} P(s_i|h_i)P(t_i|h_i)$$

as before. Where we extend the previous model is in the structure of the distribution $P(h)$. Whereas before this distribution assumed independence of the individual states, we now introduce a 1-stage Markov structure into the hidden model for M, that is we let

$$P_M(h) = P_M(h_1)P_M(h_2|h_1)\dots P_M(h_n|h_{n-1}).$$

The hidden model is illustrated in Figure 12.3. Formally, if we wish to cast this in the framework of Definition 12.5, we must define the states of the model to be

$$\{a_I\} \cup A \times \{1, \dots, n\} \cup a_F,$$

with the transition probabilities given by

$$P_M((a,i)|a_I) = \begin{cases} P_M(a) & \text{if } i = 1; \\ 0 & \text{otherwise,} \end{cases}$$

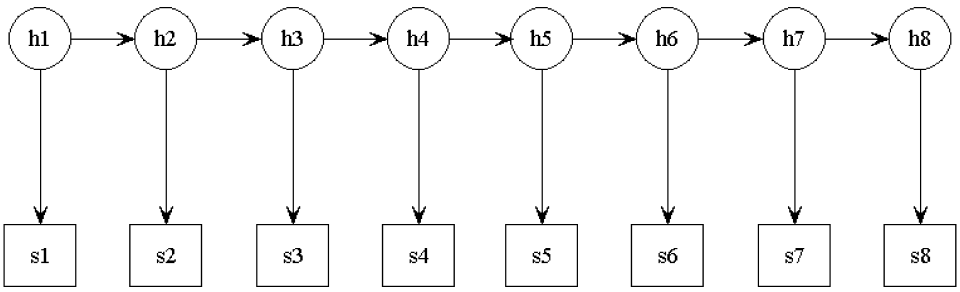

Fig. 12.3. The hidden Markov model for a fixed length string.

$$\begin{aligned} P_M\left((a,i)\,|\,(b,j)\right) &= \begin{cases} P_M\left(a|b\right) & \text{if } i=j+1; \\ 0 & \text{otherwise,} \end{cases} \\ P_M\left(a_F|\,(b,j)\right) &= \begin{cases} 1 & \text{if } i=n; \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

This means that in order to marginalise, we need to sum over a more complex probability distribution for the hidden states to obtain the corresponding marginalisation kernel

$$\begin{aligned} \kappa\left(s,t\right) &= \sum_{h\in A^n} P(s|h)P(t|h)P_M(h) \\ &= \sum_{h\in A^n}\prod_{i=1}^{n} P\left(s_i|h_i\right)P\left(t_i|h_i\right)P_M\left(h_i|h_{i-1}\right), \end{aligned} \qquad (12.2)$$

where we have used the convention that $P_M(h_1|h_0) = P_M(h_1)$.

Remark 12.7 [Describing the models] We consider each hidden sequence h as a template for the sequences s, t in the sense that if we are in state h_i at position i, the probability that the observable sequence has a symbol s_i in that position is a function of h_i. In our generative model, sequences are generated independently from the hidden template with probabilities $P(s_i|h_i)$ that can be specified by a matrix of size $|\Sigma| \times |A|$. So given this matrix and a fixed h, we can compute $P(s|h)$ and $P(t|h)$. The problem is that there are $|A|^n$ different possible models for generating our sequences s, t, that is the feature space is spanned by a basis of $|A|^n$ dimensions. Furthermore, we consider a special generating process for h of Markov type: the probability of a state depends only on the preceeding state. The consequent marginalisation step will therefore be prohibitively expensive, if performed in a direct way. As in Chapter 11, we will exploit dynamic programming techniques to speed it up. ■

Consider the set of states A_a^k of length k that end with a given by

$$A_a^k = \left\{ h \in A^k : h_k = a \right\}.$$

We introduce a series of subkernels $\kappa_{k,a}$ for $k = 1, \ldots, n$ and $a \in A$ as follows

$$\begin{aligned} \kappa_{k,a}(s,t) &= \sum_{h \in A_a^k} P(s|h)P(t|h)P_M(m) \\ &= \sum_{h \in A_a^k} \prod_{i=1}^{k} P(s_i|h_i) P(t_i|h_i) P_M(h_i|h_{i-1}), \end{aligned}$$

where we have implicitly extended the definitions of $P(s|h)$ and $P(h)$ to cover the case when h has fewer than n symbols by ignoring the rest of the string s.

Clearly, we can express the HMM kernel simply by

$$\kappa(s,t) = \sum_{a \in A} \kappa_{n,a}(s,t).$$

For $k = 1$ we have

$$\kappa_{1,a}(s,t) = P(s_1|a) P(t_1|a) P_M(a).$$

We now obtain recursive equations for computing $\kappa_{k+1,a}(s,t)$ in terms of $\kappa_{k,b}(s,t)$ for $b \in A$, as the following derivation shows

$$\begin{aligned} \kappa_{k+1,a}(s,t) &= \sum_{h \in A_a^{k+1}} \prod_{i=1}^{k+1} P(s_i|h_i) P(t_i|h_i) P_M(h_i|h_{i-1}) \\ &= \sum_{b \in A} P(s_{k+1}|a) P(t_{k+1}|a) P_M(a|b) \\ &\qquad \sum_{h \in A_b^k} \prod_{i=1}^{k} P(s_i|h_i) P(t_i|h_i) P_M(h_i|h_{i-1}) \\ &= \sum_{b \in A} P(s_{k+1}|a) P(t_{k+1}|a) P_M(a|b) \kappa_{k,b}(s,t). \end{aligned}$$

When computing these kernels we need to use the usual dynamic programming tables, one for each $\kappa_{k,b}(s,t)$, though of course we can overwrite those obtained for $k-1$ when computing $k+1$. The result is summarised in the following algorithm.

Algorithm 12.8 [Fixed length hidden Markov model kernel] The fixed length Hidden Markov Model kernel is computed in Code Fragment 12.1. ■

```
Input     Symbol strings $s$ and $t$, state transition probability matrix $P_M(a|b)$,
          initial state probabilities $P_M(a)$
          and conditional probabilities $P(\sigma|a)$ of symbols given states.

Process   Assume $p$ states, $1, \dots, p$.
          for $a = 1 : p$
              $\text{DPr}(a) = P(s_1|a) P(t_1|a) P_M(a)$;
          end
          for $i = 1 : n$
           Kern = 0;
           for $a = 1 : p$
              $\text{DP}(a) = 0$;
              for $b = 1 : p$
                $\text{DP}(a) = \text{DP}(a) + P(s_i|a) P(t_i|a) P_M(a|b) \text{DPr}(b)$;
              end
             $\text{Kern} = \text{Kern} + \text{DP}(a)$;
           end
           DPr = DP;
          end
Output    $\kappa(s,t) = \text{Kern}$
```

Code Fragment 12.1. Pseudocode for the fixed length HMM kernel.

The complexity of the kernel can be bounded from the structure of the algorithm by

$$O\left(n\,|A|^2\right).$$

Remark 12.9 [Reversing sums and products] It is possible to see the dynamic programming as partially reversing the sum and product of equation 12.2. At each stage we compute a product of the sums of the previous phase. This enable the algorithm to break down the very large sum of (12.2) into manageable stages. ■

Example 12.10 The kernel presented here compares two strings of the same length under a hidden Markov model assumption. The strings are assumed to be 'aligned', that is the generic model h assumes that the ith symbol in each sequence was generated in a conditionally independent way, given the hidden state h_i, and that the sequence of hidden states has a

Markovian structure. This makes sense in those situations in which there is no possibility of insertions or deletions, but we can accurately model random substitutions or mutations of symbols.

12.1.5 Pair hidden Markov model kernels

There are several possible ways to generalise the fixed length hidden Markov model kernel given above. In this and the next subsection we present two different extensions. The kernel presented above makes sense if we assume that two sequences have the same length and are generated by a 'template' sequence that has a Markov structure.

This subsection develops a kernel for strings of different lengths but again generated by an underlying Markov process, a case that has significant importance in computational biology. An alternative way to model the generative process of a fixed length sequence is to assume that all positions in the sequence are associated with the leaves of a tree, being generated simultaneously by an underlying evolution-like process. A generalisation to Markovian trees will be presented in the following section.

In this section we investigate the case when the two sequences have different lengths, generated by emission from hidden states that again satisfy the Markov property. This will make it possible to consider the insertion and deletion of symbols that could have created gaps in the pairwise alignment, hence creating alternative alignments between the sequences and possibly altering their lengths.

The change required to achieve this extra flexibility is at first sight very small. We simply augment the alphabet with the empty symbol ε. As a result a state can generate a symbol for one string while adding nothing to another. This means that the states will no longer correspond to the symbol positions. For this reason we must consider a general Markov model given by a set A of states with transition probabilities given by a matrix

$$M_{ab} = P_M\left(b|a\right)$$

giving the probability of moving to state b from state a. There is now no longer a linear ordering imposed on the states, which therefore can form a general probabilistic finite state automaton. We must, however, specify an initial state a_I with no associated emission and with no transitions to it. The model M comprises all possible trajectories h of states beginning at the initial state $h_0 = a_I$. It is also possible to require the trajectories to end a specified final state a_F, though to simplify the exposition we omit this restriction. It follows that the probability of a trajectory $h = (h_0, h_1, \ldots, h_N)$

is given by

$$P_M(h) = \prod_{i=1}^{N} P_M(h_i|h_{i-1}).$$

The emission probabilities for non-initial states are given by the probabilities

$$P(\sigma|a), \text{ for } a \in A \text{ and } \sigma \in \hat{\Sigma} = \Sigma \cup \{\varepsilon\},$$

with the probability that a trajectory h of length N generates a string s given by

$$P(s|h) = \sum_{\hat{s} \in \hat{\Sigma}^N} [\hat{s} = s] \prod_{i=1}^{N} P(s_i|h_i),$$

where we use $[\hat{s} = s]$ for the function outputting 1 when the two strings are equal after deletion of any empty symbols and 0 otherwise. As usual we consider the marginalisation kernel

$$\kappa(s,t) = \sum_{h \in M} P(s|h)P(t|h)P_M(h).$$

Remark 12.11 [Pair hidden Markov models] Our method of defining the generation is slightly unconventional. The more standard approach allows states to emit one symbol for each sequence, only one of which can be the empty character. Occasionally the symbol emission is also shifted from the states to the transitions. We have preferred to have emission from states to link with the previous and following subsections. Allowing states to emit pairs of characters complicates the verification of conditional independence, since we can no longer simply view the kernel as estimating the probability of the two strings arising from the modelling process, that is as a marginalisation kernel. The simplification achieved here results in some slight changes to the dynamic programme computations where we introduce pair hidden Markov models as a computational tool, but we feel that this is a price worth paying for the conceptual simplification. ■

Example 12.12 We give an example to illustrate the computations of pair HMMs. Two proteins or genes are said to be homologous if they are both descendents from a common ancestor. The sequences may differ from the ancestor's as a result of deletions, insertions and mutations of symbols where these processes occurred independently for the two sequences after their first evolutionary divergence. The distance can be estimated by computing the alignments of the two sequences. For example the two sequences s ='AACGTACTGACTA' and t ='CGTAGAATA' may be aligned in the following way

```
AACGTACTGA-CTA
--CGTA--GAA-TA
```

among others. When assessing the probability that two sequences are homologous, a meaningful measure of similarity and hence potentially an interesting kernel, is given by computing the probabilities of the two sequences being generated from a common ancestor summed over all potential ancestors. The ancestors are modelled as a sequence of states encoding the changes that occurred at each point, deletion, insertion or mutation. The state sequence is assumed to be Markovian since the probabilities of specific changes depend only on the changes occurring at the previous step. This approach typically leads to different possible 'alignments' of the sequences, each with a given probability. For each possible hidden ancestor model h we can calculate the probability that each of the two sequences is its descendent $P(s|h)$ and $P(t|h)$, and we can sum this over all possible ancestors, marginalising in this way over all possible evolutionary ancestors: $P(s,t) = \sum_h P(s|h)P(t|h)P(h)$.

The apparently small change we have introduced to handle gaps complicates the dynamic programming computations. We must now not only consider summing over all sequences ending in a certain stage, but also specify the positions we have reached in each of the strings in much the same way as we did for the string kernels in Chapter 11. Before we can do this we have an additional problem that states which emit ε can emit ε to both sequences complicating the recursion as in this case no new symbols are processed. This problem will be overcome by transforming our hidden Markov model into a so-called pair hidden Markov model generating the same distribution. We begin with a definition of pair hidden Markov models.

Definition 12.13 [Pair hidden Markov model] A data model whose states emit ordered pairs of data will be known as pair data model. Hence, a pair model M is specified by a set of distributions for $h \in M$ that assigns to each pair of data items (x_1, x_2) a probability $P_h(x_1, x_2) = P((x_1, x_2)|h)$. A pair hidden Markov model is a hidden Markov model whose states emit ordered pairs of data items. ■

Note that in general a pair data model will not define a kernel if we take

$$\kappa(x_1, x_2) = \sum_{h \in M} P((x_1, x_2)\,|h), \tag{12.3}$$

since the generation of the two examples is not necessarily independent. This will not, however, concern us since we use the pair hidden Markov model to generate a distribution that we know to be a valid kernel. Our use of the pair hidden Markov model is simply a stepping stone towards creating an efficient algorithm for evaluating that kernel.

The initial pair hidden Markov model is simply the hidden Markov model where each state uses its generating function to generate two symbols independently rather than the single symbol generated by the original model. The kernel defined by equation (12.3) is clearly the same as the marginalisation kernel by its definition.

The pair hidden Markov model is now adapted in two stages. The first creates a new model $\hat{M}$ from M by separating each state that can generate pairs $(\varepsilon, \varepsilon)$ as well as symbols from Σ into two states, one that generates only $(\varepsilon, \varepsilon)$ and one that only generates pairs with at least one symbol from Σ. The second stage involves the removal of the states that emit only $(\varepsilon, \varepsilon)$. In both cases we must verify that the distribution generated by the model remains unchanged.

For the first stage consider a state a that can emit $(\varepsilon, \varepsilon)$ as well as other combinations. We create a new state a^ε for each such state a. The state a^ε emits $(\varepsilon, \varepsilon)$ with probability 1, while the new emission probabilities $\hat{P}((\sigma_1, \sigma_2)\,|a)$ for a are given by

$$\hat{P}((\sigma_1, \sigma_2)\,|a) = \begin{cases} \frac{P(\sigma_1|a)P(\sigma_2|a)}{1-P(\varepsilon|a)^2} & \text{if } \sigma_1 \neq \varepsilon \text{ or } \sigma_2 \neq \varepsilon; \\ 0 & \text{otherwise,} \end{cases}$$

so that a can no longer emit $(\varepsilon, \varepsilon)$. Furthermore the transition probabilities are adapted as follows

$$\begin{aligned} P_{\hat{M}}(a|b) &= P_M(a|b)\left(1 - P(\varepsilon|a)^2\right) \text{ for all } b \neq a, \\ P_{\hat{M}}(a|a) &= P_{\hat{M}}(a|a^\varepsilon) = P_M(a|a)\left(1 - P(\varepsilon|a)^2\right), \\ P_{\hat{M}}(a^\varepsilon|b) &= P_M(a|b)\,P(\varepsilon|a)^2 \text{ for all } b \neq a, \\ P_{\hat{M}}(a^\varepsilon|a) &= P_{\hat{M}}(a^\varepsilon|a^\varepsilon) = P_M(a|a)\,P(\varepsilon|a)^2. \end{aligned}$$

It is readily verified that the state a^ε in $\hat{M}$ corresponds to emitting the pair $(\varepsilon, \varepsilon)$ in state a in M, and so the pair hidden Markov model $\hat{M}$ generates an identical distribution over pairs to that generated by M.

The second state of the adaptation involves removing all the states that only emit the pair $(\varepsilon, \varepsilon)$. This is a standard operation on probabilistic finite state automata involving appropriately adjusting the transition probabilities between the remaining states. We omit the details as it is not central to our development.

We therefore now assume that we have a pair hidden Markov model none of whose states emit the pair $(\varepsilon, \varepsilon)$. Consider the sequences of states A_a that end with a given by

$$A_a = \left\{ h \in \hat{M} : h_{|h|} = a, h_0 = a_I \right\}.$$

We introduce a series of subkernels $\kappa_{i,j,a}$ for $a \in A$ as follows

$$\kappa_{i,j,a}\left(s,t\right) = \sum_{h \in A_a} \hat{P}(s\left(1:i\right), t\left(1:j\right)|h) P_{\hat{M}}(h).$$

Clearly, we can express the marginalisation kernel quite simply as

$$\kappa\left(s,t\right) = \sum_{a \in A} \kappa_{|s|,|t|,a}\left(s,t\right),$$

though if a finish state were specified we would only need the corresponding subkernel. The base of the dynamic programme recursion requires us to compute $\kappa_{0,0,a}\left(s,t\right)$ for all states $a \in A$. Clearly, for a_I we have $\kappa_{0,0,a_I}\left(s,t\right) = 1$, while for general $a \in A \setminus \{a_I\}$

$$\kappa_{0,0,a}\left(s,t\right) = 0.$$

We now obtain recursive equations for computing $\kappa_{i,j,a}\left(s,t\right)$ in terms of $\kappa_{i-1,j,b}\left(s,t\right), \kappa_{i,j-1,b}\left(s,t\right)$ and $\kappa_{i-1,j-1,b}\left(s,t\right)$ for $b \in A$, as the following derivation shows

$$\begin{aligned}
\kappa_{i,j,a}\left(s,t\right) &= \sum_{h \in A_a} P((s,t)|h) P_{\hat{M}}(h) \\
&= \sum_{b \in A} P\left(\left(s_i, \varepsilon\right)|a\right) P_{\hat{M}}\left(a|b\right) \kappa_{i-1,j,b}\left(s,t\right) \\
&\quad + \sum_{b \in A} P\left(\left(\varepsilon, t_j\right)|a\right) P_{\hat{M}}\left(\hat{a}|b\right) \kappa_{i,j-1,b}\left(s,t\right) \\
&\quad + \sum_{b \in A} P\left(\left(s_i, t_j\right)|a\right) P_{\hat{M}}\left(a|b\right) \kappa_{i-1,j-1,b}\left(s,t\right).
\end{aligned}$$

This is again a set of equations for $\kappa_{i,j,a}\left(s,t\right)$ assuming that we have previously computed $\kappa_{i-1,j,b}\left(s,t\right), \kappa_{i,j-1,b}\left(s,t\right)$ and $\kappa_{i-1,j-1,b}\left(s,t\right)$ and results in the following algorithm.

Algorithm 12.14 [Pair hidden Markov models] The pair hidden Markov model kernel is computed in Code Fragment 12.2. ■

```
Input    Symbol strings $s$ and $t$, state transition probability matrix
         $P_M(a|b)$, initial state 0, final state $p$ and conditional probabilities
         $P((\sigma_1, \sigma_2)|a)$ of pairs of symbols given states.

Process  Assume $p$ states, $0, \dots, p$. Lengths of $s$ and $t$ are $n$ and $m$.
         for $a = 0 : p$
            Kern $(0, 0, a) = 0$;
            for $i = 0 : n$  Kern $(i, -1, a) = 0$;
            for $j = 0 : m$  Kern $(-1, j, a) = 0$;
         end
         Kern $(0, 0, 0) = 1$;
         for $i = 1 : n$
           for $a = 1 : p$
             Kern $(i, 0, a) = 0$;
              for $b = 0 : p$
                Kern $(i, 0, a) =$ Kern $(i, 0, a)$
                              $+P((s_i, \varepsilon)|a) P_M(a|b)$ Kern $(i - 1, 0, b)$;
          end
         end
         for $i = 0 : n$
          for $j = 1 : m$
             for $a = 1 : p$
               Kern $(i, j, a) = 0$;
                for $b = 0 : p$
                  Kern $(i, j, a) =$ Kern $(i, j, a)$
                   $+P((s_i, \varepsilon)|a) P_M(a|b)$ Kern $(i - 1, j, b)$
                     $+P((\varepsilon, t_j)|a) P_M(a|b)$ Kern $(i, j - 1, b)$
                     $+P((s_i, t_j)|a) P_M(a|b)$ Kern $(i - 1, j - 1, b)$;
          end
         end
Output   $\kappa(s, t) =$ Kern $(n, m, p)$
```

Code Fragment 12.2. Pseudocode for the pair HMM kernel.

Remark 12.15 [Defining pair HMMs directly] We have introduced pair hidden Markov models as a stepping stone towards an efficient algorithm as they enable us to remove the possibility of no symbols at all being emitted by a state. Authors typically choose to define their models using pair hidden Markov models. This has the advantage of making the design more natural in some cases. The disadvantage is that extra conditions need to be applied to ensure that a valid kernel is generated, essentially ensuring that paths between states that emit two symbols are independent whether they emit symbols to one string or the other. ■

12.1.6 Hidden tree model kernels

The previous subsection considers unequal length sequences using a hidden Markov model. In this section we return to aligned equal length sequences, but consider a more complex generating model.

Another way to model the hidden structure generating two aligned strings of the same length n is to assume their symbols are associated to the n leaves of a tree, in which the state of each node depends probabilistically on the state of its parent node. The state of the leaves is assumed to represent the sequence. This is a probabilistic graphical model, and can be used as a generative model of the sequence and so to design a marginalisation kernel

$$P_M(s,t) = \sum_{h \in M} P(s|h)P(t|h)P_M(h).$$

The set M of models is therefore defined by a fixed tree of internal nodes $v_0, v_1, \ldots, v_N$ each labelled with $h(v_i)$ from a set A of states. We will assume that the state $a = h(n)$ of a node n is compatible with the number $\text{child}(n) = \text{child}(a)$ of its children and determines the distributions $P_{i,a}(b) = P_i(b|a)$ of probability that the ith child $\text{ch}_i(v)$ is in state b given its parent is in state a. The states of leaf nodes are assumed to be the corresponding symbol from Σ. Hence, after we specify a probability distribution $P_M(a)$ over the state $h(v_0)$ of the root node, this implies a distribution over the states of the n leaf nodes $l_1, \ldots, l_n$. This probabilistic graphical model describes a distribution of probability over all sequences from Σ^n given by

$$P(s) = \sum_{h \in M} \prod_{k=1}^{n} [h(l_k) = s_k] \, P_M(h(v_0)) \prod_{i=0}^{N} \prod_{j=1}^{\text{child}(h(v_i))} P_j(h(\text{ch}_j(v_i)) \, | h(v_i)).$$

Example 12.16 Consider the case of a fixed binary tree with binary states and symbols. Furthermore, we assume the conditional probabilities are the same for both children, so that we have only two degrees of freedom ϵ and δ for the internal nodes

$$\begin{array}{ll} P(0|1) = \epsilon & P(1|0) = \delta \\ P(1|1) = 1-\epsilon & P(0|0) = 1-\delta, \end{array}$$

with another degree of freedom τ for the probability of the root-state

$$P(h(v_0) = 1) = \tau \text{ and } P(h(v_0) = 0) = 1 - \tau.$$

Hence, for this example we need only specify three parameters.

Figure 12.4 shows an example of a hidden tree model. The tree model

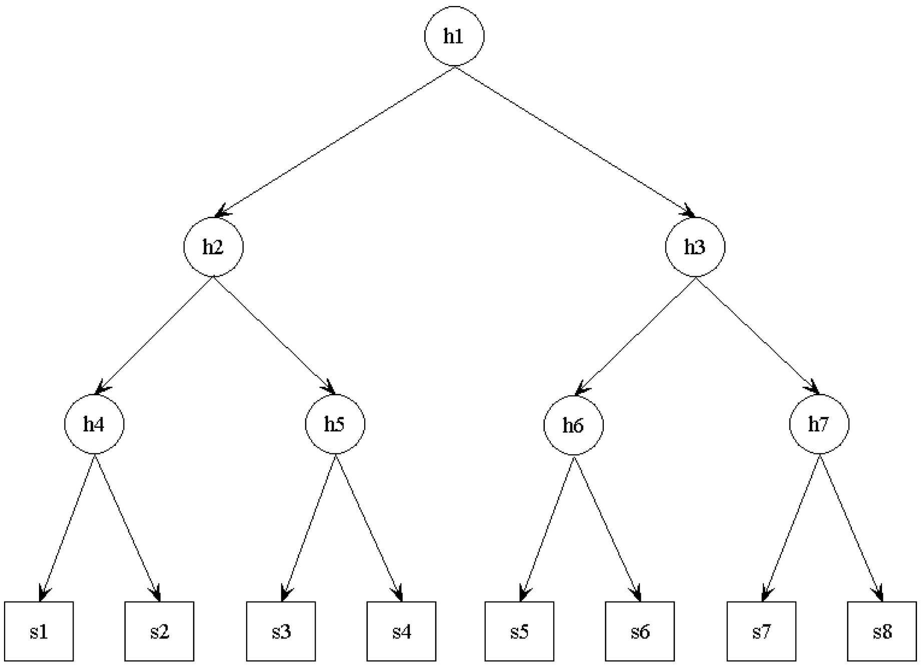

Fig. 12.4. A hidden tree model for generating a fixed length string.

is another example of a probabilistic graphical model. In general the number of different states of the tree make it impractical to perform a direct computation of the associated marginalisation kernel

$$\kappa(s,t) = \sum_{h \in M} P(s|h)P(t|h)P_M(h).$$

Since this probability is parametrised by the state of the internal nodes, there are exponentially many states h that can be used to measure the probability of a sequence corresponding to the exponentially many features $\phi_h(s) = P(s|h)$ for $h \in M$ that define the feature space. Once again we must devise a dynamic programme that can short-circuit the full computation by making use of the specific structure of the model.

This dynamic programme is an example of a message passing algorithm, that with a single depth-first traversal of the tree can compute the joint probability of the two input sequences summed over all compatible hidden states. This quantity is stored at the root node, after the traversal when the relative message passing is completed. At each node, a number of values or 'messages' are stored, and they are calculated based only on the values stored at their children, as will be described below.

Once again we must store separate values at each node for each state. As usual we define these as auxiliary kernels

$$\kappa_{v,a}(s,t) = \sum_{h \in T_a(v)} P(s|h)P(t|h)P_M(h),$$

where $T_a(v)$ is the sub-model generated on the complete subtree rooted at the node v with the node v fixed into state a, i.e. all labellings possible for the nodes of the subtree with the root in state a. The critical recursion that now makes possible the computation is as follows

$$\begin{aligned}
\kappa_{v,a}(s,t) &= \sum_{h \in T_a(v)} P(s|h)P(t|h)P_M(h) \\
&= \sum_{\mathbf{b} \in A^{\text{child}(a)}} \prod_{j=1}^{\text{child}(a)} P(b_j|a) \sum_{h \in T_{b_j}(\text{ch}_j(v))} P(s|h)P(t|h)P_M(h) \\
&= \sum_{\mathbf{b} \in A^{\text{child}(a)}} \prod_{j=1}^{\text{child}(a)} P(b_j|a)\, \kappa_{\text{ch}_j(v),b_j}(s,t) \\
&= \prod_{j=1}^{\text{child}(a)} \sum_{b \in A} P(b|a)\, \kappa_{\text{ch}_j(v),b}(s,t),
\end{aligned}$$

where the final line again follows from the use of the distributive law as in equation (12.1). If we consider the case of a binary tree with binary states, the computation at an internal node v involves two values

$$\kappa_{\text{ch}_j(v),0}(s,t) \text{ and } \kappa_{\text{ch}_j(v),1}(s,t)$$

being received from each of the two children $\text{ch}_1(v)$ and $\text{ch}_2(v)$. These four values are then combined with the probabilities $P(0|0)$, $P(0|1)$, $P(1|0)$ and $P(1|1)$ to compute the values

$$\kappa_{v,0}(s,t) \text{ and } \kappa_{v,1}(s,t)$$

subsequently passed on to its parent. In general there are $|A|$ messages from each child that must be received and appropriately processed.

The value of the kernel is given

$$\kappa(s,t) = \sum_{a \in A} P_M(a)\, \kappa_{v_o,a}(s,t),$$

that is the appropriately weighted sum of the kernels obtained by fixing the root node v_0 to each of the possible states. Similarly, the base of the recursion is given by

$$\kappa_{l_i,a}(s,t) = [s_i = t_i = a],$$

since this corresponds to a deterministic state at the leaf l_i which either agrees with both of the corresponding symbols s_i and t_i or not. The result is summarised in the following algorithm that uses a recursive function returning an array of kernel values indexed by the states.

Algorithm 12.17 [Hidden tree model kernel] The hidden tree model kernel is computed in Code Fragment 12.3. ■

The complexity of Algorithm 12.17 can be estimated by observing that each node v of the tree is visited once with a call to Treerecur (v) and that the computation involved for each call is bounded by $O\left(|A|^2 d^+\right)$, where d^+ is the maximum number of children of the tree nodes. Hence, the overall algorithm has complexity

$$O\left(N\,|A|^2\,d^+\right),$$

since N is the number of nodes in the tree.

Remark 12.18 [Swapping sums and products] Note how the strategy succeeds in breaking down the overall sum of products into a staged sequence of manageable sums, one at each internal node. Hence, there has again been a partial swapping of the sums and products from the original definition, each swap leading to a corresponding reduction in the complexity. ■

Example 12.19 Consider Example 12.16. Since this is a binary tree the recursion is somewhat simpler with all the indexing sets running from 0 to 1. For this case lines 17-26 of the algorithm can, for example, be replaced by

```
17 | Kern $(0) = ((1-\delta)\,$Kerp$\,(0,0) + \epsilon\,$Kerp$\,(0,1))$
18 |               $((1-\delta)\,$Kerp$\,(1,0) + \epsilon\,$Kerp$\,(1,1))$;
19 | Kern $(1) = (\delta\,$Kerp$\,(0,0) + (1-\epsilon)\,$Kerp$\,(0,1))$
20 |               $(\delta\,$Kerp$\,(1,0) + (1-\epsilon)\,$Kerp$\,(1,1))$;
```

Example 12.20 [Phylogenetic profiles] In its simplest form a phylogenetic profile of a gene or a protein is a string of bits each indicating the presence or absence of a homologue of the gene or protein in a different organism. It

```
Input    Symbol strings $s$ and $t$, tree structure and conditional state
         probabilities $P_M(a|b)$, initial state probabilities $P_M(a)$
--------------------------------------------------------------------------
Process  Assume $p$ states, $1, \ldots, p$.
2        DP = Treerecur $(v_0)$;
3        Kern = 0;
4        for $a = 1 : p$
5            Kern = Kern $+ P_M(a)$ DP $(a)$;
6        end
--------------------------------------------------------------------------
where    Treerecur $(v)$
8        if $v = l_i$
9          if $s_i = t_i$  DP $(s_i) = 1$;
10         return DP;
11       end
12       for $j = 1 :$ child $(v)$
13             DPp $(j, :)$ = Treerecur (ch$_j$ $(v)$);
14       for $a = 1 : p$
15        if child $(a) \neq$ child $(v)$
16           DP $(a) = 0$;
17        else
18           DP $(a) = 1$;
19           for $j = 1 :$ child $(a)$
20             DPq = 0;
21             for $b = 1 : p$
22                 DPq = DPq $+ P_M(a|b)$ DPp $(j, b)$;
23              end
24              DP $(a)$ = DP $(a)$ DPq;
25          end
26        end
27        return DP;
--------------------------------------------------------------------------
Output   $\kappa(s, t)$ = Kern
```

Code Fragment 12.3. Pseudocode for the hidden tree model kernel.

is assumed that functionally related genes have similar phylogenetic profiles. In this context the question of defining similarity between such profiles is crucial. Instead of just counting the number of organisms in which the two genes are simultaneously present or absent, intuitively one could hope to obtain more information by comparing the entire evolutionary history of the two genes. Though the precise history is not available, it can be modeled by a phylogenetic tree and probabilities can be computed with it. A phylogenetic tree of a group of organisms is a rooted tree, where each leaf represents an organism currently existing, and each internal node represents an extinct species, with the edges of the tree indicating that the child species evolved from the parent. The transmission of genes during evolution can be modelled using the tree models considered in this section, that is a probabilistic graphical model that defines a joint probability distribution P for a set of random binary variables indexed by the nodes of the tree. The probabilities that a species has a gene when its parent does not is encoded in $P(0|1)$ and so on. Hence, assuming knowledge of the phylogenetic tree, the kernel we have defined can be used to measure the relatedness of two genes based on their phylogenetic profiles.

Remark 12.21 [More general tree kernels] It is possible to extend the above kernel to consider all subtrees of the given tree structure, where the subtrees are obtained by pruning complete subtrees. Furthermore it is possible to consider the case in which we do not know the topology of the tree, and we want to marginalise over a large number of possible topologies. More information and pointers to the relevant literature can be found in Section 12.4. ■

12.2 Fisher kernels

12.2.1 From probability to geometry

The main idea behind marginalisation kernels was to consider the probability of the data given one model $P(x|m)$ as a feature. Given a single model this is very little information that is potentially misleading since two very different data items could be given the same probability by the model without this indicating any similarity. The marginalisation kernels overcame this weakness by comparing the scores of data items under several different models, typically exponentially many models.

We now consider an alternative strategy known as the Fisher kernel that attempts to extract from a single generative model more information than simply their output probability. The aim is to analyse how the score depends on the model, which aspects of the model have been important in determining that score, and hence obtaining information about the internal representation of the data items within the model.

For such an approach to work we require the single model to be more flexible so that we can measure how it adapts to individual data items. We will therefore need the model to be smoothly parametrised so that derivatives of the model with respect to the parameters can be computed.

Remark 12.22 [Relation between generative model and Fisher kernels] These ideas are clearly related. In the case of marginalisation kernels we typically create a finite but very large set of models over which the scores are computed. We now consider a parametrised model with a particular parameter setting, but measure how the model changes in the neighbourhood of that setting. Hence, we can see the approach as considering a class of models obtained by deforming the current model. In this case, however, the information is summarised in a feature space whose dimension is equal to the number of parameters. Despite these connections the approaches are complementary in the sense that marginalisation kernels typically consider a set of models indexed by a discrete set, for example the paths of a hidden Markov model. Any smooth parameters are treated as constant. The Fisher kernel on the other hand is based only on the modelling of smooth parameters. ■

Remark 12.23 [Features from internal computations] Yet another approach would be to extract information from the 'internal computations' of the model in order to evaluate its probabilities. It is the generation process of a data point that should enable a fuller description in the feature space, so that the difference in the generation process and not just in the computed probability can be exploited for comparing two examples x and z. ■

Our starting point is a smooth parametric model $P_{\boldsymbol{\theta}^0}(x) = P\left(x|\boldsymbol{\theta}^0\right)$ of the data that computes the probability of a given input x being produced by the underlying generative process with the vector of parameters $\boldsymbol{\theta}$ set to $\boldsymbol{\theta}^0$.

We will construct features ϕ_i one for each model parameter θ_i by examining slight perturbations of that parameter around the given setting θ_i^0, in this way obtaining several models or perspectives by extracting information about how the different parameters affect the score of x.

More specifically, if one wanted to adapt the parameters of a model in order to accommodate a new data item by increasing its likelihood under the model, the parameters will have to be adapted in different ways. If we use a gradient approach to parameter tuning, the partial derivatives of the score $P_{\boldsymbol{\theta}^0}(x)$ with respect to the parameters contain the information about how much each parameter should be moved in order to accommodate the new point.

In this perspective, it seems natural to compare two data points through the directions in which they 'stretch' the parameters of the model, that is by viewing the score function at the two points as a function of the parameters and comparing the two gradients. If the gradient vectors are similar it means that the two data items would adapt the model in the same way, that is from the point of view of the given parametric model at the current parameter setting they are similar in the sense that they would require similar adaptations to the parameters.

Remark 12.24 [Features corresponding to constraints] There are other intuitive interpretations of the approach of describing the data items by the perturbations of the score caused by slight parameter movements. Consider a model that calculates the probability of a data item x being generated by checking if it satisfies certain constraints, with each constraint being assessed by one parameter. If some are violated, the data item is less representative of the model and so will receive a lower score, depending on the magnitude and number of the violations. The Fisher kernel compares two data points with respect to all relevant properties of the data model by comparing the pattern of behaviour of each with respect to the different constraints. One could also view these features as the Lagrange multipliers assessing the degree to which the input is attempting to violate the different constraints. ■

What we have introduced in an intuitive way is a representation of the data that is based on a single parametrised probabilistic model. The purpose of Fisher kernels is to capture the internal representation of the data exploited by such models in order to compute its score.

Definition 12.25 [Smoothly parametrised model class] A *smoothly parametrised model class* is a set of probability distributions

$$M = \{P_{\boldsymbol{\theta}}(x) : \boldsymbol{\theta} \in \Theta\}$$

on an input space X, where $\Theta \subseteq \mathbb{R}^N$ for some finite N. We say that each parameter setting $\boldsymbol{\theta}$ determines a model $m(\boldsymbol{\theta})$ of the data. We define the *likelihood* of a data item x with respect to the model $m(\boldsymbol{\theta})$ for a given setting of the parameters $\boldsymbol{\theta}$ to be

$$\mathcal{L}_{\boldsymbol{\theta}}(x) = P(x|\boldsymbol{\theta}) = P_{\boldsymbol{\theta}}(x).$$

■

Probabilistic models assess the degree of fit of a data point to a model by its likelihood. We can learn the model parameters θ by adapting them to maximise the likelihood of the training set, i.e. the probability

$$\prod_{i=1}^{\ell} \mathcal{L}_{\boldsymbol{\theta}}(x_i)$$

that all of the training points were generated independently in the model. The problem of learning a model in this way is something with which we are not concerned in this book. We will assume throughout that the setting of the parameters $\boldsymbol{\theta}^0$ has been determined. This can be done, for example, by gradient ascent of the likelihood of the training set. From this perspective an intuitive way to assess the relationship of a new point to the model is to see what update would be necessary to the parameters in order to incorporate it, that is what is the gradient of the likelihood at this point. If a point is a prototypical example of the points accepted by the model, its likelihood will already be close to maximal and little or no tuning of the parameters will be necessary. If a point is very unusual with respect to some property, the gradient in that direction will be large in order to incorporate the new point.

As we have seen in the discussion so far the information we want to exploit is contained in the gradient of the likelihood function of the given data point taken with respect to the tunable parameters. In practice we consider the log of the likelihood function since this means that products of probabilities become sums, while at the same time the resulting gradient vector is just rescaled. We can also use the second order derivatives to refine the comparison.

Definition 12.26 [Fisher score and Fisher information matrix] The *log-likelihood* of a data item x with respect to the model $m(\boldsymbol{\theta}^0)$ for a given setting of the parameters $\boldsymbol{\theta}^0$ is defined to be

$$\log \mathcal{L}_{\boldsymbol{\theta}^0}(x).$$

Consider the vector gradient of the log-likelihood

$$\mathbf{g}\left(\boldsymbol{\theta}, x\right) = \left(\frac{\partial \log \mathcal{L}_{\boldsymbol{\theta}}(x)}{\partial \theta_i}\right)_{i=1}^{N}.$$

The *Fisher score* of a data item x with respect to the model $m(\boldsymbol{\theta}^0)$ for a given setting of the parameters $\boldsymbol{\theta}^0$ is

$$\mathbf{g}\left(\boldsymbol{\theta}^0, x\right).$$

The *Fisher information matrix* with respect to the model $m(\boldsymbol{\theta}^0)$ for a given setting of the parameters $\boldsymbol{\theta}^0$ is given by

$$\mathbf{I}_M = \mathbb{E}\left[\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)'\right],$$

where the expectation is over the generation of the data point x according to the data generating distribution. ■

The Fisher score gives us an embedding into the feature space $\mathbb{R}^N$ and hence immediately suggests a possible kernel. The matrix $\mathbf{I}_M$ can be used to define a non-standard inner product in that feature space.

Definition 12.27 [Fisher kernel] The invariant Fisher kernel with respect to the model $m(\boldsymbol{\theta}^0)$ for a given setting of the parameters $\boldsymbol{\theta}^0$ is defined as

$$\kappa(x,z) = \mathbf{g}\left(\boldsymbol{\theta}^0, x\right)' \mathbf{I}_M^{-1} \mathbf{g}\left(\boldsymbol{\theta}^0, z\right).$$

The practical Fisher kernel is defined as

$$\kappa(x,z) = \mathbf{g}\left(\boldsymbol{\theta}^0, x\right)' \mathbf{g}\left(\boldsymbol{\theta}^0, z\right).$$

■

Remark 12.28 [Problem of small norms] Notice that this kernel may give a small norm to very typical points, while atypical points may have large derivatives and so a correspondingly larger norm. The derivative of the log implies division by $P\left(x|\boldsymbol{\theta}\right)$, which will to some extent reinforce this effect. There is, therefore, a danger that the inner product between two typical points can become very small, despite their being similar in the model. This effect is can of course be overcome by normalising the kernel. ■

Remark 12.29 [Using the Gaussian kernel] Another way to use the Fisher score that does not suffer from the problem described above is to use the Gaussian kernel based on distance in the Fisher score space

$$\kappa\left(x,z\right) = \exp\left(-\frac{\left\|\mathbf{g}\left(\boldsymbol{\theta}^0, x\right) - \mathbf{g}\left(\boldsymbol{\theta}^0, z\right)\right\|^2}{2\sigma^2}\right).$$

■

We examine now some simple examples to illustrate the concepts introduced so far.

Example 12.30 [1-dim Gaussian Fisher kernel] Consider the model class M on $X = \mathbb{R}$ defined by the 1-dimensional Gaussian distributions

$$M = \left\{ P(x|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) : \boldsymbol{\theta} = (\mu, \sigma) \in \mathbb{R}^2 \right\}.$$

The log-likelihood of a point x as a function of the parameters $\boldsymbol{\theta} = (\mu, \sigma)$ is

$$\log \mathcal{L}_{(\mu,\sigma)}\left(x\right) = -\frac{(x-\mu)^2}{2\sigma^2} - \frac{1}{2}\log\left(2\pi\sigma\right).$$

Hence, the Fisher scores of a point x at parameter setting $\boldsymbol{\theta}^0 = (\mu_0, \sigma_0)$ is

$$\mathbf{g}\left(\boldsymbol{\theta}^0, x\right) = \left(\frac{(x-\mu_0)}{\sigma_0^2}, \frac{(x-\mu_0)^2}{\sigma_0^3} - \frac{1}{2\sigma_0}\right).$$

If we use the embedding

$$\boldsymbol{\phi}(x) = \mathbf{g}\left(\boldsymbol{\theta}^0, x\right),$$

as in the practical Fisher kernel, then we obtain the following embedding for the case when $\mu_0 = 0$ and $\sigma_0 = 1$

$$\boldsymbol{\phi}(x) = \left(x, x^2 - 0.5\right),$$

which is equivalent to the polynomial kernel of degree 2 with the real line being embedded onto the curve $y = x^2 - 0.5$ as shown in Figure 12.5. Using

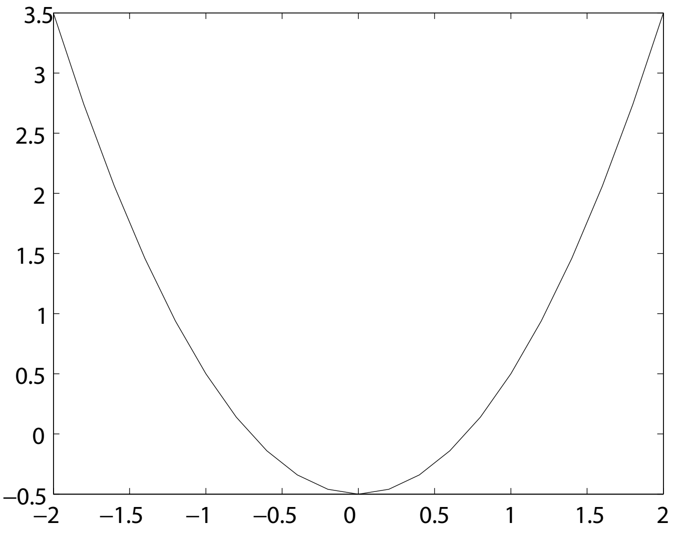

Fig. 12.5. Feature space embedding corresponding to the Fisher kernel of a one-dimensional Gaussians.

hyperplanes in this 2-dimensional space shows that the embedding will allow classifications that reflect exclusions of independent amounts of each tail of the Gaussian, something not possible with the standard (1-dimensional) inner product. Note that this will be true even if the true Gaussian is different from that given by the parameter setting $\boldsymbol{\theta}^0$, though the closer to the true distribution it is the easier it will be to achieve the separation.

Example 12.31 [Two fixed width Gaussians] If we assume our data has been generated by two Gaussians with pre-fixed widths both equal to σ and means stored as the parameters $\boldsymbol{\theta} = (\mu^+, \mu^-)$, the gradient of the log-likelihood with respect to the two tunable parameters is

$$\mathbf{g}\left(\boldsymbol{\theta}^0, x\right) = \left(\frac{\left(x - \mu_0^+\right)}{\sigma^2}, \frac{\left(x - \mu_0^-\right)}{\sigma^2}\right).$$

This will not enrich the capacity of the function class since it maps to a line in the *two*-dimensional feature space.

The two examples above illustrate how different assumptions on the source generating the data can change the embedding map, and a correct or even approximately correct guess of the model can improve the embedding.

We now show with examples how Fisher kernels can be used to deal with structured objects such as for example the strings of symbols in text or DNA. Frequently a generative model exists for such objects or can be inferred from easily available unlabelled data. We will gradually move from artificial to real cases, starting from fixed length strings of independently generated symbols, moving to Markov models and finally to HMMs. As before with exponential case, the same string can be embedded in different spaces depending on the generative model we decide to use. Finally we will also discuss an application to the case of text, where the generative model involves the use of latent variables in the role of 'topics'.

Example 12.32 [Markov strings] We consider a generative model for strings of length n parametrised by probabilities $p_{u\to\sigma}$ that the next character following a string u of length k is σ. We therefore compute the likelihood of a string s as

$$\prod_{j=1}^{n-k} p_{s(j:j+k-1)\to s_{j+k}},$$

that is the product of the probabilities for each of the characters that have at least k predecessors in the string. If we assume that

$$\sum_{\sigma\in\Sigma} p_{u\to\sigma} = 1, \text{ for all } u \in \Sigma^k,$$

and that the first k symbols of the strings are fixed (if necessary these could be k special marker symbols added to the beginning of the string), the likelihoods corresponds to a distribution over the set of possible strings. This implies certain constraints on the parameters $p_{u\to\sigma}$ which would affect the calculation of the derivatives. We therefore choose to parametrise the model with arbitrary real-values $\boldsymbol{\theta} = (a_{u\to\sigma})_{u\in\Sigma^k,\sigma\in\Sigma}$ with

$$p_{u\to\sigma} = \frac{a_{u\to\sigma}}{\sum_{\sigma\in\Sigma} a_{u\to\sigma}}.$$

The Fisher score now becomes

$$\begin{aligned}
\mathbf{g}\left(\boldsymbol{\theta}^0, s\right)_{u\to\sigma} &= \frac{\partial \log \mathcal{L}_{\boldsymbol{\theta}}(s)}{\partial a_{u\to\sigma}} \\
&= \sum_{j=1}^{n-k} \frac{\partial \log p_{s(j:j+k-1)\to s_{j+k}}}{\partial a_{u\to\sigma}} \\
&= \sum_{j=1}^{n-k} [s\,(j:j+k-1) = u] \left([s_{j+k} = \sigma] \frac{1}{a_{s(j:j+k-1)\to s_{j+k}}} \right. \\
&\qquad \left. - \frac{1}{\sum_{\sigma\in\Sigma} a_{s(j:j+k-1)\to\sigma}} \right).
\end{aligned}$$

If we consider the parameter setting $\boldsymbol{\theta}^0$ for which all $a_{u\to\sigma} = p_{u\to\sigma} = |\Sigma|^{-1}$ the computation simplifies to

$$\mathbf{g}\left(\boldsymbol{\theta}^0, s\right)_{u\to\sigma} = \left|\{(v_1, v_2) : s = v_1 u\sigma v_2\}\right| |\Sigma| - \left|\{(v_1, v_2) : s = v_1 u v_2\}\right|.$$

Ignoring the second term this becomes the p-spectrum kernel of length $k+1$ scaled by $|\Sigma|$. The second term subtracts the first term for the given u averaged in s over the possible next characters σ, hence projecting the subset of features that include a k-string u to the hyperplane perpendicular to the all ones vector.

Remark 12.33 [Arbitrary length strings] Example 12.32 can be extended to arbitrary length strings by introducing an end-of-string state to which we can transfer with certain probability at any time. More generally the approach suggests that we may improve the performance of the p-spectrum kernel by tuning the parameters $a_{u\to\sigma}$ for example using counts of the transitions in the whole corpus. This corresponds to learning the model from the training data.

Note how the states of the corresponding Markov model are labelled by subsequences $u\sigma$ of length $k+1$, with the transitions defined by adding the next symbol to the suffix of length k. This suggests that we could generalise the approach by allowing states that correspond to different length subsequences perhaps chosen so that they are neither too frequent or too rare. The transition for a state u and symbol σ would be to the longest suffix of $u\sigma$ that corresponds to a state. Again the transition parameters could be inferred from the training data. ■

The examples we have considered so far have all been for the practical Fisher kernel. The invariant Fisher kernel suffers from the drawback that it requires us to compute a random matrix

$$\mathbf{I}_M = \mathbb{E}\left[\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)'\right]$$

where x is drawn according to the input distribution, something that we are not able to do in practice. Before considering this problem we will first indicate why the name 'invariant' is appropriate. If we reparametrise the model using an invertible differentiable transformation

$$\boldsymbol{\psi} = \boldsymbol{\psi}\left(\boldsymbol{\theta}\right),$$

we might expect that the resulting kernel would change. The embedding will certainly alter and so the practical Fisher kernel will change with it, but the invariant Fisher kernel is not affected by this reparametrisation since if $\tilde{\kappa}$ is the transformed kernel we have

$$\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)' = \left(\frac{\partial \log \mathcal{L}_{\boldsymbol{\psi}}(x)}{\partial \psi_i}\right)_{i=1}^{N} \mathbf{J}\left(\boldsymbol{\psi}\right) = \mathbf{g}\left(\boldsymbol{\psi}^0, x\right)' \mathbf{J}\left(\boldsymbol{\psi}^0\right),$$

where $\mathbf{J}\left(\boldsymbol{\psi}^0\right)$ is the Jacobian of the transformation $\boldsymbol{\psi}$ evaluated at $\boldsymbol{\psi}^0$. It follows that

$$\begin{aligned}
&\tilde{\kappa}(x_1, x_2) \\
&\quad = \mathbf{g}\left(\boldsymbol{\psi}^0, x_1\right)' \tilde{\mathbf{I}}_M^{-1} \mathbf{g}\left(\boldsymbol{\psi}^0, x_2\right) \\
&\quad = \mathbf{g}\left(\boldsymbol{\theta}^0, x_1\right)' \mathbf{J}\left(\boldsymbol{\psi}^0\right)^{-1} \\
&\qquad \mathbb{E}\left[\mathbf{J}\left(\boldsymbol{\psi}^0\right)'^{-1} \mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)' \mathbf{J}\left(\boldsymbol{\psi}^0\right)^{-1}\right]^{-1} \mathbf{J}\left(\boldsymbol{\psi}^0\right)'^{-1} \mathbf{g}\left(\boldsymbol{\theta}^0, x_2\right) \\
&\quad = \mathbf{g}\left(\boldsymbol{\theta}^0, x_1\right)' \mathbf{J}\left(\boldsymbol{\psi}^0\right)^{-1} \mathbf{J}\left(\boldsymbol{\psi}^0\right) \\
&\qquad \mathbb{E}\left[\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)'\right]^{-1} \mathbf{J}\left(\boldsymbol{\psi}^0\right)' \mathbf{J}\left(\boldsymbol{\psi}^0\right)'^{-1} \mathbf{g}\left(\boldsymbol{\theta}^0, x_2\right) \\
&\quad = \mathbf{g}\left(\boldsymbol{\theta}^0, x_1\right)' \mathbb{E}\left[\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)'\right]^{-1} \mathbf{g}\left(\boldsymbol{\theta}^0, x_2\right) = \kappa\left(x_1, x_2\right).
\end{aligned}$$

The invariant Fisher kernel is indeed invariant to any smooth invertible reparametrisation of the model. This would appear to be a desirable property if we believe that the choice of parametrisation is somewhat arbitrary. We will see, however, that this view may be misleading.

Let us return to the question of computing the Fisher information matrix

$$\mathbf{I}_M = \mathbb{E}\left[\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)'\right].$$

A natural approximation for the matrix is to take the empirical rather than the true expectation. In other words we estimate the matrix by averaging over the sample

$$\hat{\mathbf{I}}_M = \hat{\mathbb{E}}\left[\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x\right)'\right] = \frac{1}{\ell}\sum_{i=1}^{\ell}\mathbf{g}\left(\boldsymbol{\theta}^0, x_i\right)\mathbf{g}\left(\boldsymbol{\theta}^0, x_i\right)'.$$

In this case $\hat{\mathbf{I}}_M$ is simply the covariance matrix of the Fisher scores. Hence, the Fisher kernel becomes equivalent to whitening these scores (see Algorithm 6.16), with the associated dangers of amplifying noise if certain parameters are not relevant for the information contained in the training set. Hence, the invariance comes at the cost of possibly reducing the signal to noise ratio in the representation.

12.2.2 Fisher kernels for hidden markov models

We now consider deriving Fisher kernels for the hidden Markov models that we developed to create generative kernels. Consider first the fixed length hidden Markov models described in Section 12.1.4. We now view the model as the sum over all of the state paths or individual models with the parameters the various transition and emission probabilities, so that for a particular parameter setting the probability of a sequence s is given by

$$P_M\left(s\right) = \sum_{m\in A^n} P(s|m)P_M(m) = \sum_{m\in A^n} P_M(s,m),$$

where

$$P_M(m) = P_M(m_1)P_M(m_2|m_1)\dots P_M(m_n|m_{n-1}),$$

and

$$P(s|m) = \prod_{i=1}^{n} P\left(s_i|m_i\right)$$

so that

$$P_M(s,m) = \prod_{i=1}^{n} P\left(s_i|m_i\right)P\left(m_i|m_{i-1}\right).$$

The parameters of the model are the emission probabilities $P(s_i|m_i)$ and the transition probabilities $P_M(m_i|m_{i-1})$. For convenience we introduce parameters

$$\theta_{s_i|m_i} = P(s_i|m_i) \text{ and } \tau_{m_i|m_{i-1}} = P_M(m_i|m_{i-1}),$$

where we use the convention that $P_M(m_1) = P_M(m_1|m_0)$ with $m_0 = a_0$ for a special fixed state $a_0 \notin A$. We again have the difficulty experienced in Example 12.32 that these parameters are not independent. Using the same approach as was adopted in that example we introduce the unconstrained parameters

$$\theta_{\sigma,a} \text{ and } \tau_{a,b}$$

with

$$\theta_{\sigma|a} = \frac{\theta_{\sigma,a}}{\sum_{\sigma'\in\Sigma}\theta_{\sigma',a}} \text{ and } \tau_{a|b} = \frac{\tau_{a,b}}{\sum_{a'\in A}\tau_{a',b}}.$$

We assemble these values into a parameter vector $\boldsymbol{\theta}$. Furthermore we assume that the parameter setting at which the derivatives are computed satisfies

$$\sum_{\sigma\in\Sigma}\theta_{\sigma,a} = \sum_{a\in A}\tau_{a,b} = 1, \tag{12.4}$$

for all $a, b \in A$ in order to simplify the calculations.

We must compute the derivatives of the log-likelihood with respect to the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\tau}$. The computations for both sets of parameters follow an identical pattern, so to simplify the presentation we first derive a template that subsumes both cases. Let

$$\bar{\psi}(b,a) = \frac{\psi(b,a)}{\sum_{b'\in B}\psi(b',a)}, \text{ for } a \in A \text{ and } b \in B.$$

Let

$$Q(\mathbf{a},\mathbf{b}) = \prod_{i=1}^{n}\bar{\psi}(b_i,a_i)\,c_i,$$

for some constants c_i. Consider the derivative of $Q(\mathbf{a},\mathbf{b})$ with respect to the parameter $\psi(b,a)$ at a point $\left(\mathbf{a}^0,\mathbf{b}^0\right)$ where

$$\sum_{b\in B}\psi\left(b,a_i^0\right) = 1 \text{ for all } i. \tag{12.5}$$

We have

$$\frac{\partial Q(\mathbf{a},\mathbf{b})}{\partial\psi(b,a)}$$

$$
\begin{aligned}
&= \sum_{k=1}^{n} c_k \prod_{i \neq k} \bar{\psi}\left(b_i^0, a_i^0\right) c_i \frac{\partial}{\partial \psi(b,a)} \frac{\psi(b_k, a_k)}{\sum_{b' \in B} \psi(b', a_k)} \\
&= \sum_{k=1}^{n} \left(\frac{[b_k^0 = b][a_k^0 = a]}{\sum_{b' \in B} \psi\left(b', a_k^0\right)} - \frac{\psi\left(b_k^0, a_k^0\right)[a_k^0 = a]}{\left(\sum_{b' \in B} \psi\left(b', a_k^0\right)\right)^2} \right) c_k \prod_{i \neq k} \bar{\psi}\left(b_i^0, a_i^0\right) c_i \\
&= \sum_{k=1}^{n} \left(\frac{[b_k^0 = b][a_k^0 = a]}{\bar{\psi}(b,a)} - [a_k^0 = a] \right) \prod_{i = k} \bar{\psi}\left(b_i^0, a_i^0\right) c_i \\
&= \sum_{k=1}^{n} \left(\frac{[b_k^0 = b][a_k^0 = a]}{\bar{\psi}(b,a)} - [a_k^0 = a] \right) Q\left(\mathbf{a}^0, \mathbf{b}^0\right),
\end{aligned}
$$

where we have made use (12.5) to obtain the third line from the second. We now return to considering the derivatives of the log-likelihood, first with respect to the parameter $\theta_{\sigma,a}$

$$
\begin{aligned}
\frac{\partial \log P_M(s|\boldsymbol{\theta})}{\partial \theta_{\sigma,a}} &= \frac{1}{P_M(s|\boldsymbol{\theta})} \frac{\partial}{\partial \theta_{\sigma,a}} \sum_{m \in A^n} \prod_{i=1}^{n} P(s_i|m_i) P_M(m_i|m_{i-1}) \\
&= \frac{1}{P_M(s|\boldsymbol{\theta})} \sum_{m \in A^n} \frac{\partial}{\partial \theta_{\sigma,a}} \prod_{i=1}^{n} \frac{\theta_{s_i,m_i}}{\sum_{\sigma \in \Sigma} \theta_{\sigma,m_i}} \tau_{m_i|m_{i-1}}.
\end{aligned}
$$

Letting $\mathbf{a}$ be the sequence of states m and $\mathbf{b}$ the string s, with $\psi(a,b) = \theta_{b,a}$ and $c_i = \tau_{m_i|m_{i-1}}$ we have

$$
Q(\mathbf{a}, \mathbf{b}) = \prod_{i=1}^{n} \frac{\theta_{s_i,m_i}}{\sum_{\sigma \in \Sigma} \theta_{\sigma,m_i}} \tau_{m_i|m_{i-1}} = P_M(s, m|\boldsymbol{\theta}).
$$

It follows from the derivative of Q that

$$
\begin{aligned}
\frac{\partial \log P_M(s|\boldsymbol{\theta})}{\partial \theta_{\sigma,a}} &= \sum_{m \in A^n} \sum_{k=1}^{n} \left(\frac{[s_k = \sigma][m_k = a]}{\theta_{\sigma|a}} - [m_k = a] \right) \frac{P_M(s, m|\boldsymbol{\theta})}{P_M(s|\boldsymbol{\theta})} \\
&= \sum_{k=1}^{n} \sum_{m \in A^n} \left(\frac{[s_k = \sigma][m_k = a]}{\theta_{\sigma|a}} - [m_k = a] \right) P_M(m|s, \boldsymbol{\theta}) \\
&= \sum_{k=1}^{n} \mathbb{E}\left[\frac{[s_k = \sigma][m_k = a]}{\theta_{\sigma|a}} - [m_k = a] \middle| s, \boldsymbol{\theta} \right] \\
&= \frac{1}{\theta_{\sigma|a}} \sum_{k=1}^{n} \mathbb{E}\left[[s_k = \sigma][m_k = a] | s, \boldsymbol{\theta}\right] - \sum_{k=1}^{n} \mathbb{E}\left[[m_k = a] | s, \boldsymbol{\theta}\right],
\end{aligned}
$$

where the expectations are over the hidden states that generate s. Now consider the derivatives with respect to the parameter $\tau_{a,b}$

$$\begin{aligned}\frac{\partial \log P_M(s|\boldsymbol{\theta})}{\partial \tau_{a,b}} &= \frac{1}{P_M(s|\boldsymbol{\theta})}\frac{\partial}{\partial \tau_{a,b}} \sum_{m\in A^n} \prod_{i=1}^{n} P(s_i|m_i) P_M(m_i|m_{i-1}) \\ &= \frac{1}{P_M(s|\boldsymbol{\theta})} \sum_{m\in A^n} \frac{\partial}{\partial \tau_{a,b}} \prod_{i=1}^{n} \frac{\theta_{s_i,m_i}}{\sum_{\sigma\in\Sigma}\theta_{\sigma,m_i}} \tau_{m_i|m_{i-1}}.\end{aligned}$$

Letting **a** and **b** be the sequence of states m and **b** be the same sequence of states shifted one position, with $\psi(a,b) = \tau_{a,b}$ and $c_i = \theta_{s_i|m_i}$, we have

$$Q(\mathbf{a},\mathbf{b}) = \prod_{i=1}^{n} \theta_{s_i|m_i} \frac{\tau_{m_i,m_{i-1}}}{\sum_{a'\in A}\tau_{a',m_{i-1}}} \tau_{m_i|m_{i-1}} = P_M(s,m|\boldsymbol{\theta}).$$

It follows from the derivative of Q that

$$\begin{aligned}&\frac{\partial \log P_M(s|\boldsymbol{\theta})}{\partial \tau_{a,b}} \\ &= \sum_{k=1}^{n} \sum_{m\in A^n} \left(\frac{[m_{k-1}=b][m_k=a]}{\tau_{a|b}} - [m_k=a] \right) P_M(m|s,\boldsymbol{\theta}) \\ &= \frac{1}{\tau_{a|b}} \sum_{k=1}^{n} \mathbb{E}\left[[m_{k-1}=b][m_k=a]|s,\boldsymbol{\theta}\right] - \sum_{k=1}^{n} \mathbb{E}\left[[m_k=a]|s,\boldsymbol{\theta}\right],\end{aligned}$$

It remains to compute the expectations in each of the sums. These are the expectations that the particular emissions and transitions occurred in the generation of the string s.

The computation of these quantities will rely on an algorithm known as the forwards–backwards algorithm. As the name suggests this is a two-stage algorithm that computes the quantities

$$f_a(i) = P(s_1 \ldots s_i, m_i = a),$$

in other words the probability that the ith hidden state is a with the prefix of the string s together with the probability $P(s)$ of the sequence. Following this the backwards algorithm computes

$$b_a(i) = P(s_{i+1} \ldots s_n | m_i = a).$$

Once these values have been computed it is possible to evaluate the expectation

$$\begin{aligned}&\mathbb{E}\left[[s_k=\sigma][m_k=a]|s\right] \\ &\quad = P(s_k=\sigma, m_k=a|s)\end{aligned}$$

$$= [s_k = \sigma] \frac{P(s_{k+1} \dots s_n | m_k = a) P(s_1 \dots s_k, m_k = a)}{P(s)}$$
$$= [s_k = \sigma] \frac{f_a(k) b_a(k)}{P(s)}.$$

Similarly

$$\begin{aligned}\mathbb{E}[[m_k = a]|s] &= P(m_k = a|s) \\ &= \frac{P(s_{k+1} \dots s_n | m_k = a) P(s_1 \dots s_k, m_k = a)}{P(s)} \\ &= \frac{f_a(k) b_a(k)}{P(s)}.\end{aligned}$$

Finally, for the second pair of expectations the only tricky evaluation is $\mathbb{E}[[m_{k-1} = b][m_k = a]|s]$, which equals

$$\frac{P(s_{k+1} \dots s_n | m_k = a) P(s_1 \dots s_{k-1}, m_{k-1} = b) P(a|b) P(s_k | m_k = a)}{P(s)}$$
$$= \frac{f_b(k-1) b_a(k) \tau_{a|b} \theta_{s_k|a}}{P(s)}.$$

Hence, the Fisher scores can be evaluated based on the results of the forwards–backwards algorithm. The forwards–backwards algorithm again uses a dynamic programming approach based on the recursion

$$f_b(i+1) = \theta_{s_{i+1}|b} \sum_{a \in A} f_a(i) \tau_{b|a},$$

with $f_{a_0}(0) = 1$ and $f_a(0) = 0$, for $a = a_0$. Once the forward recursion is complete we have

$$P(s) = \sum_{a \in A} f_a(n).$$

The initialisation for the backward algorithm is

$$b_a(n) = 1$$

with the recursion

$$b_b(i) = \sum_{a \in A} \tau_{a|b} \theta_{\sigma_{i+1}|a} b_a(i+1).$$

Putting all of these observations together we obtain the following algorithm, where we only evaluate the Fisher scores for the emission probabilities for brevity.

Algorithm 12.34 [Fixed length Markov model Fisher kernel] The Fisher scores for the fixed length Markov model Fisher kernel are computed in Code Fragment 12.4. ■

```
Input    Symbol string $s$, state transition probability matrix $P_M(a|b)$,
         initial state probabilities $P_M(a) = P_M(a|a_0)$
         and conditional probabilities $P(\sigma|a)$ of symbols given states.

Process  Assume $p$ states, $0, 1, \ldots, p$.
         score $(:, :) = 0$;
         forw $(:, 0) = 0$;
         back $(:, n) = 1$;
         $f(0, 0) = 1$; Prob $= 0$;
         for $i = 1 : n$
          for $a = 1 : p$
             forw $(a, i) = 0$;
             for $b = 1 : p$
               forw $(a, i) =$ forw $(a, i) + P_M(a|b)$ forw $(b, i - 1)$;
             end
            forw $(a, i) =$ forw $(a, i)\, P(s_i|a)$;
          end
         end
         for $a = 1 : p$
           Prob $=$ Prob $+$ forw $(a, n)$;
         end
         for $i = n - 1 : 1$
            for $a = 1 : p$
               back $(a, i) = 0$;
               for $b = 1 : p$
                  back $(a, i) =$ back $(a, i) + P_M(b|a)\, P(s_{i+1}|b)$ back $(b, i + 1)$;
           end
           score $(a, s_i) =$ score $(a, s_i) +$ back $(a, i)$ forw $(a, i)$ / $(P(s_i|a)$ Prob$)$;
           for $\sigma \in \Sigma$
              score $(a, \sigma) =$ score $(a, \sigma) +$ back $(a, i)$ forw $(a, i)$ / Prob;
Output   Fisher score = score
```

Code Fragment 12.4. Pseudocode to compute the Fisher scores for the fixed length Markov model Fisher kernel.

12.3 Summary

- P-kernels arise from probability distributions over pairs of examples.
- marginalisation kernels compute the probability that two examples are generated over a class of models.
- marginalisation kernels can be computed efficiently for a number of com-

plex generative models including fixed length hidden Markov models, pair hidden Markov models and hidden tree models.
- Fisher kernels use the gradient of the log-likelihood as a feature vector.
- Fisher kernels can be efficiently evaluated for hidden Markov models.

12.4 Further reading and advanced topics

The notion of P-kernel was introduced by Haussler [52], while Watkins [155], [154] independently discussed conditional independence kernels, and the case of hidden Markov models. These ideas were later applied by Vert, defining first a family of kernels for strings based on simple probabilistic models [145] and then based on the hidden tree model [146] (our discussion of the hidden tree model is based on the work of Vert, while our discussion of the pair HMM kernel is derived from the work of Watkins, but more generally draws from the literature on context trees and on message-passing methods for marginalising probabilities in graphical models). The formal presentation of the pair-HMMs kernel is however based on the Durbin *et al.* 'forward' algorithm [42].

Pair HMMs have also been used as kernels by J.-P. Vert *et al.* [148]. Extensions of these ideas can be found in the papers of marginalised kernels of Koji Tsuda [137], [76], which also exploit stochastic context free grammars as generative models.

Another extension from trees to certain graphs is also possible using the techniques from [131], but overall the general problem is best attacked with the algorithms and concepts developed within the field of probabilistic graphical models, where the sum-product algorithm and its extension are used for marginalisation. A new book on this topic will be forthcoming shortly [68].

The notion of Fisher kernel was defined by Jaakkola and Haussler [61], [62] and later used by several authors, for example Hoffmann in [57]. For the treatment of string kernels as Fisher kernels, see [114].

Rational Kernels [25] are based on the definition of transducers. They can be related to pair HMMs but as with our presentation here have the advantage of automatically having the finitely positive semi-definite property. Furthermore, they define a very broad class of kernels. A more recent paper of Cuturi and Vert [36] also discusses string kernels based on probabilistic models and pair HMMs.

For constantly updated pointers to online literature and free software see the book's companion website: www.kernel-methods.net