Repeated Observation Models

Avi Pfeffer
Division of Engineering and Applied Sciences
Harvard University
avi@eecs.harvard.edu

Abstract

Repetition is an important phenomenon in a variety of domains, such as music, computer programs and architectural drawings. A generative model for these domains should account for the possibility of repetition. We present repeated observation models (ROMs), a framework for modeling sequences that explicitly allows for repetition. In a ROM, an element is either generated by copying a previous element, or by using a base model. We show how to build ROMs using n-grams and hidden Markov models as the base model. We also describe an extension of ROMs in which entire subsequences are repeated together. Results from a music modeling domain show that ROMs can lead to dramatic improvement in predictive ability.

Introduction

Repetition is an important phenomenon in many artifacts. It is absolutely essential to music. Much of the art of composition is reusing material in different ways. Quite often, the same rhythm will be repeated with different pitches. In other cases pitch patterns will be repeated and transformed. A good framework for modeling music will take these phenomena into account.

Repetition also appears naturally in other domains. Consider, for example, the task of modeling computer programs. The same identifiers tend to be used again and again within a function. Architectural drawings present another example. The same window design might appear several times within a room, and in many rooms in a building. Text may also contain an element of repetition, with the same names and keywords (such as “repetition”) appearing throughout a document.

Existing probabilistic frameworks for modeling sequences of elements, such as hidden Markov models (RJ86), do not take repetition into account. In this paper, we present *repeated observation models (ROMs)*, a probabilistic modeling framework that explicitly allows for repetition. ROMs are not a single model but rather an idea that can be applied to other frameworks in order to yield a new kind of model. We present two instantiations of this idea: the repetition n-gram and the repetition HMM. We also present experimental results from a music modeling domain showing the dramatic improvements in predictive power that can be obtained through the use of ROMs.

In the basic ROMs, single elements are repeated one at a time. A more realistic model would allow entire subsequences to be modeled as a whole. We develop an extension of ROMs that allows for repetition of subsequences. Our experimental results show that this extension leads to further improvements in predictive capability.

Repeated Observation Models

The basic idea of ROMs is simple. Given a generative model G that generates elements of a sequence, we produce a new generative model G' as follows: G' generates elements one by one. As elements are generated, a memory is maintained of how many times each element has appeared. Each time a new element is to be generated, with probability ρ it is generated from memory, and with probability $1-\rho$ it is generated according to the base model G. The parameter ρ is called the *memory rate* of the model.

There is one exception to the above generation process. When the very first element of a sequence is to be generated, the memory is empty, so reusing an item from memory is impossible. Therefore the first element of a sequence is generated directly according to G.

Let $P^m(o_k \mid o_{1:k-1})$ denote the probability that o_k will be generated from a memory containing $o_{1:k-1}$. This memory model is fixed in advance and not learned. There are a number of ways to implement P^m. The simplest approach for generating an element at time k is to choose a time index uniformly from 1 to $k-1$, and to produce the element at that time index. Thus the probability that an element is produced is proportional to the number of times the element appears in the memory. An alternative model would hold that more recent items are more likely to be reproduced than items from the distant past. One way to capture this is with a linear model. When generating element o_k, the probability that element o_j, where $j < k$, will be chosen is proportional to $a + bj$. Other alternatives such as an exponential model can also be used.

An important issue, whatever memory model is chosen, is its implementation. In a naive implementation, each time we want to compute $P^m(o_k \mid o_{1:k-1})$ we will examine the sequence $o_{1:k-1}$. If we compute P^m for all time points up to time S, where S is the length of the sequence, we obtain an

$O(S^2)$ algorithm. A better solution is to precompute P^m for all time points using dynamic programming, obtaining an $O(S)$ algorithm. This can be accomplished for the constant memory model, in which all we need to know is the number of times an element has appeared, not the exact points at which it has appeared. It can also be accomplished easily for the linear model, by storing the total weight associated with each possible element as the sequence is traversed.

It is important to emphasize that the observed variables do indeed need to be observed in a ROM. If the repeating variables are hidden, inference becomes extremely difficult and approximation methods are needed, because one has to sum over all possible values of the repeating elements. Therefore in this paper we assume that the elements of the sequence are always observed, hence the name "repeated observation model". Furthermore, we restrict ourselves to discrete observations. While one could imagine an approximate notion of repetition for continuous elements, exact repetition would be a very rare occurence. We now present two examples of RMs.

Repetition n-grams

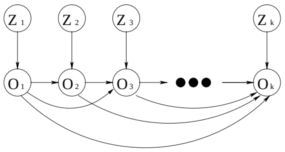

Figure 1: Dependency model for repetition n-gram

We begin with repetition n-grams. Figure 1 shows the dependency model in the form of a Bayesian network. We see that the k-th observation o_k depends on all the previous $k-1$ observations, because of generation from memory. We also introduce a variable z_k, which is true if o_k is generated from memory, false if it is generated according to the n-gram probabilities. The probability that z_k is true is ρ, the memory rate of the model, for all $k > 1$.

In using ROMs, like other probabilistic sequence models, we need a number of algorithms. The most basic algorithm computes the probability of an observation sequence $o_{1:T}$. Let $P^n(o_k \mid o_{k-n+1:k-1})$ denote the n-gram probability of o_k given the previous $n-1$ observations. Then for $k > 1$,

$$P(o_k \mid o_{1:k-1}) = \begin{array}{l} \rho P^m(o_k \mid o_{1:k-1}) + \\ (1-\rho) P^n(o_k \mid o_{k-1:k-n+1}) \end{array}$$

$P(o_1)$ is obtained from the unigram model. The probability of an observation sequence is simply the product $\prod_{k=1}^{T} P(o_k \mid o_{1:k-1})$.

The next algorithm computes the most likely assignment to hidden variables given an observation sequence. While n-grams do not have hidden variables, repetition n-grams do, namely the z_k. To find the most likely assignment to the z_k, we note from Figure 1 that the z_k are conditionally independent of each other given the observation sequence. Therefore we can choose the most likely value of each z_k independently. To start with, z_1 is always false. For $k > 1$, we compute

$$\begin{array}{l} P(z_k = T, o_k \mid o_{1:k-1}) = \rho P^m(o_k \mid o_{1:k-1}), \\ P(z_k = F, o_k \mid o_{1:k-1}) = (1-\rho) P^n(o_k \mid o_{k-1:k-n+1}) \end{array}$$

and choose the z_k that results in higher probability.

Finally, we need an algorithm to learn the values of the parameters from data. In a repetition n-gram the parameters are the memory rate ρ and the n-gram parameters. A simple version of expectation maximization (EM) (DLR77) will do the trick. In the expectation step, we compute $p_k^i = P(z_k^i = T \mid o_{1:S}^i)$ for each time point k in each training sequence i. This is obtained simply by normalizing the results of the above two equations. In the maximization step, we estimate ρ to be the average over all time points > 1 in all sequences of p_k^i. Meanwhile, the n-gram parameters are estimated by counting as usual, except that the n-gram ending with o_k^i only contributes a count of $1 - p_k^i$.

Repetition HMM

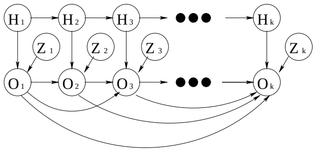

Figure 2: Dependency model for repetition HMM

The repetition HMM is a more powerful model, allowing for a hidden state that varies over time. As in an ordinary HMM, the generative model produces a Markov chain of hidden states, where the initial state is determined by the initial model $P^1(h)$ and the transition probabilities are determined by the transition model $P(h_2 \mid h_1)$. At each point an observation is generated. With probability ρ it is generated from memory, and with probability $1-\rho$ it is generated using the observation model $P(o \mid h)$. The Bayesian network structure for the repetition HMM, shown in Figure 2, appears quite dense. The hidden states h_k form a Markov chain as in ordinary HMMs. Unlike ordinary HMMs, however, each observation depends not only on its hidden state but also on the previous observations. The graph is thus highly multiply connected. However, the graph hides a lot of additional structure in the model in the form of context-specific independence (CSI) (CBK96). CSI is a situation in which two variables are independent of each other given some values of a third variable and not others. In our case, the context variables are the z_k. If z_k is true, then o_k depends only on the previous observations and not on its hidden state. If z_k is false, o_k depends only on the hidden state h_k. Thus in no circumstances does an observation depend both on the hidden state and its previous observations.

The first algorithm computes the probability of a sequence, and is analogous to the Forward-Backward algorithm for HMMs (RJ86). We present the backward version of the algorithm here. It is similar in principle, but requires some modifications to the HMM algorithm. The algorithm computes $f_k(h_k, z_k) = P(o_{k:S} \mid h_k, z_k, o_{1:k-1})$ at each time step. Note that unlike the HMM algorithm, all previous observations are included as conditioning factors for the future observations. This is because it is no longer the case that future observations are independent of past ones given the hidden state. The computation proceeds separately for the cases $z_k = T$ and $z_k = F$. For $z_k = T$,

$$\begin{array}{c} f_k(h_k, T) = P(o_{k:S} \mid h_k, z_k = T, o_{1:k-1}) \\ = \\ P(o_k \mid h_k, z_k = T, o_{1:k-1}) P(o_{k+1:S} \mid h_k, z_k = T, o_{1:k}) \\ = \\ \sum_{h_{k+1}, z_{k+1}} \left(\begin{array}{l} P^m(o_k \mid o_{1:k-1}) \times \\ P(h_{k+1} \mid h_k, z_k = T, o_{1:k}) \times \\ P(z_{k+1} \mid h_k, z_k = T, h_{k+1}, o_{1:k}) \times \\ P(o_{k+1:S} \mid h_k, z_k = T, h_{k+1}, z_{k+1}, o_{1:k}) \end{array} \right) \\ = \\ P^m(o_k \mid o_{1:k-1}) \times \\ \sum_{h_{k+1}, z_{k+1}} P(h_{k+1} \mid h_k) P(z_{k+1}) f_{k+1}(h_{k+1}, z_{k+1}) \end{array}$$

For the case where $z_k = F$, we get the similar result of

$$\begin{array}{l} f_k(h_k, F) = P(o_k \mid h_k) \times \\ \quad \sum_{h_{k+1}, z_{k+1}} P(h_{k+1} \mid h_k) P(z_{k+1}) f_{k+1}(h_{k+1}, z_{k+1}) \end{array}$$

These expressions for f form the basis of a dynamic programming algorithm. The interesting point to note is that the second step of the derivation makes use of the CSI properties of the model. This results in two different possible terms at the front of the expression, each of them simple. The dynamic programming algorithm begins by setting $f_S(h_S, T) = P^m(o_S \mid o_{1:S-1})$, and $f_S(h_S, F) = P(o_S \mid h_S)$. In looping backwards, care must be taken at the first time step, in which an element cannot be selected from memory, so $f_1(h_1, T)$ is undefined. Therefore we only compute $f_1(h_1, F)$ at the first time step. We can then return $P(o_{1:S}) = \sum_{h_1} P^1(h_1) f_1(h_1, F)$.

We skip the algorithm for computing the most likely explanation of an observation sequence, which uses similar ideas to the previous algorithm. Instead we move on to the algorithm for learning the parameters of a repetition HMM from data. As for repetition n-grams, this algorithm is a variant of EM, and follows along similar lines to the Baum-Welch algorithm. As always, the algorithm iterates between an Expectation step, in which it computes the values of expected sufficient statistics, and a Maximization step in which it reoptimizes the parameters of the model. The expected statistics to be computed are as follows:

$$\begin{array}{l} E[N_h] = \sum_{i=1}^{M} \sum_{k=1}^{S^i} P(h_k^i = h \mid o_{1:S^i}^i) \\ E[N_{h_1 h_2}] = \sum_{i=1}^{M} \sum_{k=2}^{S^i} P(h_{k-1}^i = h_1, h_k^i = h_2 \mid o_{1:S^i}^i) \\ E[N_h^F] = \sum_{i=1}^{M} \sum_{k=1}^{S^i} P(h_k^i = h, z_k^i = F \mid o_{1:S^i}^i) \\ E[N_{ho}^F] = \sum_{i=1}^{M} \sum_{k=1}^{S^i} P(h_k^i = h, o_k^i = o, z_k^i = F \mid o_{1:S^i}^i) \\ E[N_z] = \sum_{i=1}^{M} \sum_{k=2}^{S^i} P(z_k^i = z \mid o_{1:S^i}^i) \\ E[N_h^1] = \sum_{i=1}^{M} P(h_1^i = h \mid o_{1:S^i}^i) \end{array}$$

Note that instead of computing the number of times observation o appears in hidden state h, we compute the number of times that it does so when the generation is not from memory. It is only when the generation is not from memory that we can learn from the coappearance of h and o. Once we have estimated these statistics, the maximization step is as one would expect. The only thing to note is that when computing ρ, we only consider time points after the first, so the denominator uses $S^i - 1$:

$$\begin{array}{cc} P(h_2 \mid h_1) = \frac{E[N_{h_1 h_2}]}{E[N_{h_1}]} & P(o \mid h) = \frac{E[N_{ho}^F]}{E[N_h^F]} \\ P^1(h) = \frac{E[N_h^1]}{M} & \rho = \frac{E[N_T]}{\sum_{i=1}^{M} (S^i - 1)} \end{array}$$

It remains to explain how to compute the expected sufficient statistics. Here, like Baum-Welch, we use our version of the backwards algorithm as a subroutine to compute $f_k(h_k, z_k)$ for all time points, ending with the first. We can then iterate forwards to compute the probabilities needed. The iteration forwards computes $P(h_k \mid o_{1:S})$ at each time step. The computation uses the fact that

$$P(h_{k-1}, h_k, z_k \mid o_{1:S}) = \frac{1}{Z} \begin{array}{l} P(h_{k-1} \mid o_{1:S}) P(h_k \mid h_{k-1}) \\ P(z_k) f_k(h_k, z_k) \end{array}$$

where Z is a normalizing factor. Once we have computed the joint probability of h_{k-1}, h_k and z_k, using previously computed terms, we can use marginalization to obtain the probabilities we need for learning.

Repeating Subsequences

Up to this point, we have considered models of repetition in which single elements are retrieved from memory and reused, one at a time. An alternative model is to have subsequences of elements retrieved from memory and reused in their entirety. This may be a more realistic model for applications. In music, for example, it is quite natural to repeat a four-measure unit, rather than repeat each individual measure separately.

In order to model repetition of subsequences, we introduce two hidden variables. z_k indicates the number of remaining elements of a subsequence to be copied at point k. In particular, if z_k is non-zero, o_k is generated from memory, whereas if z_k is zero, o_k is generated according to the underlying model. w_k indicates the position in the sequence from which o_k is derived, if o_k is generated from memory. For example, if o_k is 2 and w_k is 8, $o_k = o_8$ and $o_{k+1} = o_9$. When $z_k = 0$, w_k is immaterial; by convention, it is set to 0.

The dynamic model of z and w is simple. If $z_k > 1$, then deterministically $z_{k+1} = z_k - 1$ and $w_{k+1} = w_k + 1$. If on the other hand $z_k = 1$ or 0, we must make a new decision at time point $k+1$ as to whether o_k is generated from memory, and if so, how long a subsequence is copied. This is achieved by selecting z_{k+1} at random from a multinomial distribution over 0 and possible subsequence lengths. The probability of selecting a particular z_{k+1} will be denoted $\rho(z_{k+1})$, extending the notation from before. If z_{k+1} is non-zero, w_{k+1} is drawn uniformly from the integers 1 to k.

An interesting point to note is that the sequence being repeated can overlap with the sequence being generated. For example, if z_{k+1} is chosen to be 2, while w_{k+1} is k, then the sequence of length 2 beginning at k is repeated beginning at point $k + 1$, with an overlap at point $k + 1$. The net result in this situation is that o_{k+1} will be equal to o_k, while o_{k+2} will be equal to o_{k+1} and therefore also to o_k. This is a useful feature. Situations where the same small group repeats itself several times in succession are quite common. It is advantageous to model them with a single long repetition with high explanatory power, rather than with several shorter repetitions.

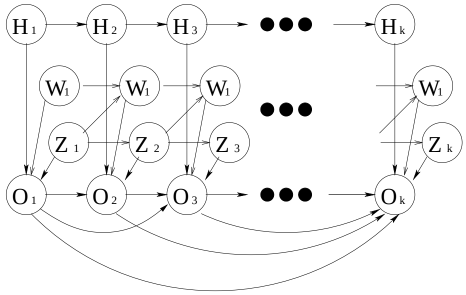

Figure 3: Dependency model for subsequence HMM

We explain the technical details of subsequence repetition models using HMMs as the underlying model. The Bayesian network structure for this model is shown in Figure 3. As before, there is a good deal of structure that is not shown in the figure. z_k and w_k act together as a multiplexer, so that when they are known, o_k either depends on h_k or is equal to a *single* previous observation. Furthermore, w_{k+1} only depends on w_k when $z_k > 1$, in which case w_k deterministically sets w_{k+1}, while z_k deterministically sets Z_{k+1}. Meanwhile, if z_k is 0 or 1, we do not care which as far as w_{k+1} and z_{k+1} are concerned.

One approach to reasoning with this model is to create a hidden state that consists of h_k, z_k, w_k. In a backward pass, we would compute $f(h_k, z_k, w_k) = P(o_{k:S} \mid h_k, z_k, w_k)$. This would be computed by summing over all possible $h_{k+1}, z_{k+1}, w_{k+1}$, using the probability of transitioning from h_k, z_k, w_k to $h_{k+1}, z_{k+1}, w_{k+1}$. A subsequent forward pass would proceed similarly, computing $P(h_k, z_k, w_k \mid o_{1:S})$ at each point.

The problem with this approach is that in both the backward and forward passes, the cost for each time step is quadratic in the number of hidden states. If we let L denote the maximum allowed repetition length, H the number of hidden states of the underlying HMM, and S the length of the sequence, which determines the range of w_k, the total cost over all S steps of the sequence is $O(H^2L^2S^3)$. Since S is often quite large, this is unacceptable.

We address this issue using two optimizations. The first is to notice that f and P are quite sparse. In order for $f(h_k, z_k, w_k)$ to be positive, when $z_k > 0$, it must be the case that $o_{w_k} = o_k$. It must further be the case that there is a subsequence of length z_k beginning at w_k that matches a corresponding sequence beginning at k. We therefore precompute, for each point in the sequence, all values of z_k and w_k that have these properties. These values are stored in a set $\mathbf{R}_k$. Let B be the maximum number of values in any $\mathbf{R}_k$. It will usually be the case that B is a lot smaller than LS, and the inference will cost $O(H^2B^2S)$, which is much better.

The second optimization comes from the observation that we do not need to consider all possible transitions from h_k, z_k, w_k to $h_{k+1}, z_{k+1}, w_{k+1}$, since many of them are impossible. In combination with the CSI noted above, this leads to a more efficient computation in both the backward and forward passes. We begin by breaking up the computation of f into cases, depending on the value of z_k.

$$\begin{aligned} &f_k(h_k, 0, 0) = P(o_k \mid h_k) \textstyle\sum_{h_{k+1}} \sum_{z_{k+1}, w_{k+1} \in \mathbf{R}_{k+1}} \\ &P(h_{k+1} \mid h_k)\rho(z_{k+1})P(w_{k+1} \mid z_{k+1}) f_{k+1}(h_{k+1}, z_{k+1}, w_{k+1}) \end{aligned}$$

Here $P(w_{k+1} \mid z_{k+1})$ is $1/k$ if $z_{k+1} > 0$, otherwise it assigns probability 1 to $w_{k+1} = 0$. Meanwhile, when $z_k = 1$, and $z_k, w_k \in \mathbf{R}_k$, we get the same expression, except that we no longer need $P(o_k \mid h_k)$:

$$\begin{aligned} &f_k(h_k, 1, w_k) = \textstyle\sum_{h_{k+1}} \sum_{z_{k+1}, w_{k+1} \in \mathbf{R}_{k+1}} \\ &P(h_{k+1} \mid h_k)\rho(z_{k+1})P(w_{k+1} \mid z_{k+1}) f_{k+1}(h_{k+1}, z_{k+1}, w_{k+1}) \end{aligned}$$

Finally, when $z_k > 1$ and $z_k, w_k \in \mathbf{R}_k$, we do not need to sum over z_{k+1} and w_{k+1}, and we do not need to include $\rho(z_{k+1})$ or $P(w_{k+1} \mid z_{k+1})$ since z_{k+1} and w_{k+1} are fully determined. We get

$$f_k(h_k, z_k, w_k) = \sum_{h_{k+1}} P(h_{k+1} \mid h_k) f_{k+1}(h_{k+1}, z_k{-}1, w_k{+}1)$$

Now we can unroll the computation of f_k into two separate loops, each costing $O(H^2B)$.

```
Forall $h_k$
   total1 = 0
   Forall $h_{k+1}$
      Forall $z_{k+1}, w_{k+1} \in \mathbf{R}_{k+1}$
         total1 $+= P(h_{k+1} \mid h_k)\rho(z_{k+1})P(w_{k+1} \mid z_{k+1})$
                   $f_{k+1}(h_{k+1}, z_{k+1}, w_{k+1})$
   $f(h_k, 0, 0) = P(o_k \mid h_k) *$ total1
   Forall $1, w_k \in \mathbf{R}_k$
      $f(h_k, 1, w_k) =$ total1
Forall $h_k$
   Forall $z_k, w_k \in \mathbf{R}_k$ where $z_k > 1$
      total2 = 0
      Forall $h_{k+1}$
         total2 $+= P(h_{k+1} \mid h_k) f_{k+1}(h_{k+1}, z_k - 1, w_k + 1)$
      $f(h_k, z_k, w_k) =$ total2
```

Similar techniques can be applied to the forward pass, to reduce the cost of computing $P(H_k, W_k, Z_k \mid o_{1:S})$ to $O(H^2B)$ at each time point. Thus the total cost of inference is $O(H^2BS)$. It should be pointed out that despite the optimizations, there is an extra factor of B compared to ordinary HMMs. Even though B is much smaller than LS, it may still be reasonably large, making inference with subsequence HMMs considerably slower than for ordinary HMMs or repetition HMMs.

Results

The models and algorithms in this paper were tested on a musical corpus. The corpus consisted of minuets and trios from Mozart's mature string quartets. Each piece was segmented into measures, and the rhythm of the first violin part was recorded for each measure. Thus the time points were measures, and the observations were the rhythm patterns of each measure. The rhythm of a measure was represented as a sequence of durations. Since the smallest rhythmic value appearing in the corpus was a thirty-second note, durations were represented as multiples of thirty-second note values. For example, in a measure consisting of two quarter notes followed by two eight notes, the observation was the sequence 8 8 4 4. There were twenty sequences in the corpus, ranging in length from 25 to 93 measures.

Each of the experiments described here used twenty-fold cross-validation. In each experiment the model was trained on nineteen of the sequences, and the perplexity of the remaining sequence was measured. The perplexity of a sequence under a model is 2^p, where p is the mean negative log probability, according to the model, of each observation in the sequence. Intuitively, the perplexity is the number of elements from which each observation is chosen. A lower perplexity corresponds to better predictive power of the model. In each of the following results, the geometric mean of the perplexities for the twenty sequences is reported.

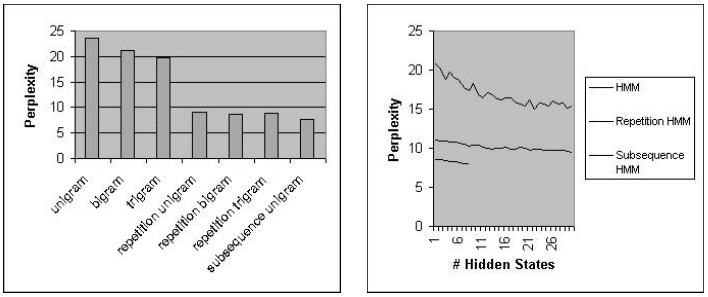

Figure 4: (a) N-gram results (b) HMM results

Figure 4(a) shows the results for the different kinds of n-gram. First, for the ordinary n-gram, we see slight improvement from unigrams to bigrams to trigrams. This indicates that the preceding two observations are not particularly useful elements for predicting the next observation. When we turn to repetition n-grams there is a dramatic improvement, going from 19.84 for the trigram to 9 for the repetition unigram. However there is almost no improvement from the repetition unigram to the repetition bigram and trigram. This suggests that for ordinary n-grams, all the structure captured by bigrams and trigrams related to immediate repetition of elements. Since this structure is already captured by the repetition unigram, there is nothing left for the repetition bigram and trigram to capture.

When we move on to subsequence repetition, there is an improvement over the basic ROMs, from 9 to 7.62. However, the worst perplexity among the twenty runs for both models was about the same. It appears that in this corpus, modeling repeating subsequences is a useful idea, but it only produces benefits in some sequences.

The results for HMMs, shown in Figure 4(b), are similar. There is a dramatic improvement for the repetition HMM over the ordinary HMM, and the performance of the subsequence HMM is even better. While adding hidden states helps the ordinary HMM, we find only a slight improvement from increasing the number of hidden states for the repetition HMM. This suggests, again, that the hidden state in the ordinary HMM was being used to capture repetitions. Despite the optimizations described in the previous section, the subsequence repetition HMM took significantly longer to run than the other methods, so a smaller number of experiments were performed. Without the optimizations, however, the method would have been completely infeasible.

Finally, we investigated whether the model by which elements are retrieved from memory makes a difference. All previous results used a model in which elements were selected uniformly. We also tried models in which recent items were more likely to be selected, using the linear model. The results are almost exactly the same as for the uniform model. This shows that, at least in this domain, there is no support for the hypothesis that recent elements are more likely to be repeated.

Related Work

ROMs bear a superficial resemblance to the Chinese Restaurant Process (CRP), recently applied to learning topic hierarchies (BGJT03). The CRP models a process in which a stream of diners arrives at a restaurant. When a new diner arrives, with some probability she will be seated at an already occupied table, otherwise she will sit at an unoccupied table. The result of this process is a distribution over partitions of integers.

There are a number of fundamental differences between the CRP and ROMs. First, the CRP is not associated with time series. Second, the CRP provides a static probability distribution over partitions. It characterizes how entities are likely to be clumped, but does not attempt to model the entities themselves and how they are generated. Third, the CRP is a self-contained model, while the ROM is an idea that can be attached to existing temporal models.

There have been many extensions to HMMs allowing elements to depend on previous elements.. For example Hierarchical HMMs (FST98) model domains with hierarchical structure and allow dependencies at multiple time scales. Mixed-memory Markov models (SJ99) allow an observation to depend on a number of previous observations. None of the previous extensions of HMMs handle repetition in the manner described in this paper.

Compression algorithms do take advantage of repeated elements in the data. Sequitur (NMW97), for example, is an algorithm that discovers hierarchical structure in sequences by identifying repeated elements. However, compression algorithms are designed to find structure in existing sequences. ROMs, in contrast, are generative models that are able to predict unseen sequences. They make it possible to exploit repetition within the framework of temporal prediction models.

Several authors have applied temporal probabilistic models to problems in music. Cemgil and Kappen (CK03) and Raphael (Rap02) have both applied graphical models to the problems of tempo induction and rhythmic parsing. Vitaniemi et al. (VKE03) have applied a simple probabilistic model of pitch to automated transcription of music. Raphael and Stoddard (RS03) and Sheh and Ellis (SE03) recently applied HMMs to harmonic analysis. Kapanci and Pfeffer (KP03) used HMMs modified to incorporate musical knowledge to model musical rhythm. None of these authors modeled reoccurence of elements, and all of these applications could benefit from ROMs.

Conclusion and Future Work

Repetition is an important feature of many domains. By making it explicit, repeated observation models are able to achieve greater accuracy when predicting the next element of a sequence, in a domain with repetition. We have shown how to incorporate repetition into n-grams and HMMs, and demonstrated that the improvement in prediction can be dramatic. We have also described how to model repeating subsequences, and shown how doing so can lead to further improvements.

Once we consider explicit models of repetition, a variety of extensions come to mind. One possibility is a repetition grammar. Here repetition would be possible at every level of the hierarchy. Whenever we want to expand a non-terminal, we might choose a subtree that has previously appeared for that non-terminal. A repetition grammar would be useful, for example, in modeling computer programs. Another possible extension is a two-dimensional repetition model. Here one would define the memory at a point to be the subset of the plane that is below and to the left of the point.

A deeper and more ambitious extension is to recognize that in many cases elements are not repeated exactly, but are varied and transformed. This extension would require a probabilistic model of the transformation process. Once we have integrated transformations into ROMs, we will be able to perform a deep parsing of sequences such as melodies, to determine not only their hierarchical structure, but how each element is derived from previous elements.

References

D.M. Blei, T.L. Griffiths, M.I. Jordan, and J.B. Tenenbaum. Hierarchical topic models and the nested Chinese restaurant process. In *Advances in Neural Information Processing Systems 16*, 2003.

M. Goldszmidt C. Boutilier, N. Friedman and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 1996.

A.T. Cemgil and B. Kappen. Monte Carlo methods for tempo tracking and rhythm quantization. *Journal of Artificial Intelligence Research*, 18:45–81, 2003.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39(1), 1977.

Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32, 1998.

E. Kapanci and A. Pfeffer. Learning style-specific rhythmic structures. In *International Computer Music Conference*, 2003.

C.G. Nevill-Manning and I.H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

C. Raphael. A hybrid graphical model for rhythmic parsing. *Artificial Intelligence*, 137(1–2):217–238, 2002.

L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE Acoustics, Speech and Signal Processing Magazine*, 3:4–16, 1986.

C. Raphael and J. Stoddard. Harmonic analysis with probabilistic graphical models. In *Fourth International Conference on Music Information Retrieval*, 2003.

A. Sheh and D.P.W. Ellis. Chord segmentation and recognition using EM-trained hidden Markov models. In *Fourth International Conference on Music Information Retrieval*, 2003.

Larry Saul and Michael Jordan. Mixed memory Markov models: Decomposing complex stochastic processes as mixture of simpler ones. *Machine Learning*, 37(1):75–87, 1999.

T. Viitaniemi, A. Klapuri, and A. Eronen. A probabilistic model for the transcription of single-voice melodies. In *Finnish Signal Processing Symposium*, 2003.