Chapter VII

Information-Theoretic Methods for Prediction in the Wireless and Wired Web

Dimitrios Katsaros, Aristotle University, Greece

Abstract

Discrete sequence modeling and prediction is an important goal and challenge for Web environments, both wired and wireless. Web clients' data-request forecasting and mobile location tracking in wireless cellular networks are characteristic application areas of sequence prediction in such environments. Accurate data-request prediction results in effective data prefetching, which combined with a caching mechanism can reduce user-perceived latencies as well as server and network loads. Also, effective solutions to the mobility tracking and prediction problem can reduce the update and paging costs, freeing the network from excessive signaling traffic. Therefore, sequence prediction comprises a very important study and development area. This chapter presents information-theoretic techniques for discrete sequence prediction. It surveys, classifies, and compares the state-of-the-art solutions, suggesting routes for further research by discussing the critical issues and challenges of prediction in wired and wireless networks.

Introduction

The proliferation of wireless cellular networks and the penetration of Internet services are changing many aspects of Web computing. Constantly increasing client populations utilize diverse devices to access the wired and wireless medium, and various heterogeneous applications (e.g., traffic- and weather-condition broadcasting, streaming video) are being developed to satisfy the eager requirements of the clients. In this environment, seamless and ubiquitous connectivity as well as low client-perceived latencies are two fundamental goals. The first goal calls for smart techniques for determining the current and future location of a mobile node, and the second goal calls for efficient and effective techniques for deducing future client requests for information pieces.

Both of the aforementioned problems are related to the ability of the underlying network to record, learn, and subsequently predict the mobile user's behavior, that is, its movements or its information needs. The success of the prediction is presupposed and is boosted by the fact that mobile users exhibit some degree of regularity in their movement and/or in their access patterns (Bhattacharya & Das, 2002; Nanopoulos, Katsaros, & Manolopoulos, 2003). This regularity may be apparent in the behavior of each individual client or in client groups. The detection of regularity patterns can lead to drastic improvements on the underlying wireless network's performance. Accurate data-request prediction results in effective data prefetching (Nanopoulos et al.) combined with a caching mechanism (Katsaros & Manolopoulos, 2004; Vakali, 2001) can reduce user-perceived latencies as well as server and network loads. Also, effective solutions to the mobility tracking and prediction problem can reduce the update and paging costs, freeing the network from excessive signaling traffic (Bhattacharya & Das).

These issues had been treated in isolation, but pioneering works (Bhattacharya & Das, 2002; Vitter & Krishnan, 1996) are paving the way for treating both problems in a homogeneous fashion. They exhibited the possibility of using methods that have traditionally been used for data compression (thus characterized as information-theoretic) in carrying out prediction. The unifying principle is that they model the respective state space as finite alphabets comprised of discrete symbols. In the mobility tracking scenario, the alphabet consists of all possible sites (cells) where the client has ever visited or might visit (assuming that the number of cells in the coverage area is finite). In the request-prediction scenario, the alphabet consists of all the data objects requested by the client plus the objects that might be requested in the future (assuming that the objects come from a database and thus their number is finite).

A smart network can record the movement (request) history and then construct a mobility (data-access) model for its clients. The history refers to the past, but the model is probabilistic and extends to the future. As uncertainty is inherent in mobile movements or requests, we can consider the requests to be the outcome of an underlying stochastic process, which can be modeled using established information-theoretic concepts and tools (Misra, Roy, & Das, 2004; Vitter & Krishnan, 1996).

In our earlier work, reported in Nanopoulos et al. (2003), we described a framework that was able to embrace some algorithms that had been presented in the context of Web prefetching. That framework was able to present those algorithms as variations of the standard PPM (prediction by partial match) technique. The emphasis of the framework was on differentiating between the techniques based on whether they record contiguous subsequences or noncontiguous subsequences. From that work, it was clear that the discovery of noncontigu-

ous subsequences required considerable computational effort and could be realized only through off-line algorithms.

Extending this work, this chapter provides a unifying framework for all the methods, which deal with the issues of location tracking and prediction and request forecasting using known information-theoretic structures, not only the PPM structures; the framework treats them as (variable or fixed-length) Markov chains and presents the different families of methods, categorizing the state-of-the-art algorithms into their respective families. It mainly deals with the discovery of contiguous subsequences, although it can relatively easily be extended to include noncontiguous subsequences. An important objective of the chapter is to include in the presentation not only the algorithms that are familiar in the wireless-communications community, but also techniques that have been developed in other disciplines, like computational biology, machine learning, and the World Wide Web, in order to achieve cross-discipline understanding and the proliferation of ideas. The purpose of the categorization is to reveal the shortcomings and advantages of each method and to identify routes for further research. Closely related to our work is that reported in Begleiter, El-Yaniv, and Yolan (2004), which, although it has a more narrow scope, examining only online prediction methods, it gives a completely different notion for the variable-length Markov chain, defining it as a combination of various Markov chains that are of different length.

The rest of the chapter is organized as follows. The next section describes in mathematical terminology the problem of discrete sequence prediction. Then the chapter surveys the different families of Markov predictors, and then provides a qualitative comparison of them. Next, we present a new online prediction algorithm that does not belong to any of the families, though it combines many of the merits presented by each family. Finally, we discuss some fields for further research, and then conclude the chapter.

The Discrete Sequence Prediction Problem

In quantifying the utility of the past in predicting the future, a formal definition of the problem is needed, which we provide in the following lines (Feder, Merhav, & Gutman, 1992; Merhav & Feder, 1998). Let $\mathbf{\Sigma}$ be an alphabet consisting of a finite number of symbols s_1, s_2, …, $s_{|\Sigma|}$, where $|\cdot|$ stands for the length or cardinality of its argument. A predictor, which is an algorithm used to generate prediction models, accumulates sequences of the type $\boldsymbol{\alpha}_i=\alpha_i^1$, α_i^2, …, α_i^{ni}, where $\alpha_i^j \in \mathbf{\Sigma}$ for all i, j and n_i denotes the number of symbols comprising $\boldsymbol{\alpha}_i$. Without loss of generality, we can assume that all the knowledge of the predictor consists of a single sequence $\boldsymbol{\alpha}= \alpha^1, \alpha^2, \ldots, \alpha^n$. Based on $\boldsymbol{\alpha}_i$, the predictor's goal is to construct a model that assigns probabilities for any future outcome given some past information. Using the characterization of the mobility or request model as a stochastic process $(X_t)_{t \in N}$, we can formulate the aforementioned goal as follows.

Definition 1 (Discrete Sequence Prediction Problem). At any given time instance t (meaning that t symbols x_t, x_{t-1}, ..., x_1 have appeared, in reverse order), calculate the conditional probability:

$$\overline{P}[X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots]$$

where $x_i \in \Sigma$ for all $x_{t+1} \in \Sigma$. This model introduces a stationary Markov chain since the probabilities are not time dependent. The outcome of the predictor is a ranking of the symbols according to their $\overline{P}$. The predictors that use such kind of prediction models are termed Markov predictors.

Depending on the application, the predictor may return only the symbol(s) with the highest probability, that is, implementing a most-probable prediction policy, or it may return the symbols with the *m* highest probabilities, that is, implementing a top-*m* prediction policy, where *m* is an administratively set parameter. In any case, the selection of the policy is a minor issue and will not be considered in this chapter, which is only concerned with methods for inferring the ranking.

The history x_t, x_{t-1}, ... used in the above definition is called the context of the predictor, and it refers to the portion of the past that influences the next outcome. The history's length (also, called the length, memory, or order of the Markov chain or predictor) will be denoted by l. Therefore, a predictor that exploits l past symbol will calculate conditional probabilities of the form:

$$\overline{P}[X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_{t-l+1} = x_{t-l+1}]$$

Some Markov predictors fix, in advance of the model creation, the value of l, presetting it in a constant k in order to reduce the size and complexity of the prediction model. These predictors and the respective Markov chains are termed fixed-length Markov chains or predictors of order k. Therefore, they compute probabilities of the form:

$$\overline{P}[X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_{t-k+1} = x_{t-k+1}]$$

where k is a constant. Although it is a nice model from a probabilistic point of view, these Markov chains are not very appropriate from the estimation point of view. Their main limitation is related to their structural poverty since there is no means to set an optimized value for k.

Other Markov predictors deviate from the fixed-memory assumption (Buhlmann & Wyner, 1999) and allow the order of the predictor to be of variable length, that is, to be a function of the values from the past.

$$\overline{P}[X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_{t-l+1} = x_{t-l+1}]$$

where $l=l(x_t,x_{t-1},...)$.

These predictors are termed variable-length Markov chains; the length l might range from

1 to *t*. If $l=l(x_t,x_{t-1},...) \equiv k$ for all x_t, x_{t-1}, ..., then we obtain the fixed-length Markov chain. The variable-length Markov predictors may or may not impose an upper bound on the considered length. The concept of variable memory offers richness in the prediction model and the ability to adjust itself to the data distribution. If we can choose in a data-driven way the function $l=l(\cdot)$, then we can only gain with respect to the ordinary fixed-length Markov chains, but this is not a straightforward problem.

The Markov predictors (fixed or variable length) base their probability calculations $\overline{P}$ on counts of the number of appearances of symbols after contexts. They also take special care to deal with the cases of unobserved symbols (i.e., symbols with zero appearance counts after contexts), assigning to them some minimum probability mass, which is acquired from the respective mass of the symbols already seen. For the location-tracking and request-prediction applications, though, the predictors usually adopt a nonprediction approach for the unobserved events and do not apply any smoothing mechanism because the possible alternative symbols may be quite large. Therefore, for the rest of the chapter, we will not deal with the zero-frequency problem and will not adopt smoothing in the presented examples.

Families of Markov Predictors

We explained earlier how Markov predictors create probabilistic models for their input sequence(s). To realize these models, they need a data structure, a dictionary to keep track of the contexts of interest, and some counts used in the calculation of the conditional probabilities $\overline{P}$. The preferred choice for this task is the use of digital search trees (trees). The root node of the tree corresponds to the null event or symbol, whereas every other node of the tree corresponds to a sequence of events; the sequence is used to label the node. An invariant for the trees is that no two children of a father node may have the same label. In the rest of the chapter, we will consider a Markov predictor to be equivalent to its respective tree. Each node is accompanied by a counter, which depicts how many times this event has appeared after the sequence of events corresponding to the path from the root to the node's father that has been observed.

For our convenience, we present some definitions useful for the sequel of the chapter. We use the sample sequence of events $\boldsymbol{\alpha}$=aabacbbabbacbbc. The length of $\boldsymbol{\alpha}$ is the number of symbols it contains, that is, $|\boldsymbol{\alpha}|=15$. We term that the maximal prefix of a (sub)sequence, say, acb, is the (sub)sequence without its rightmost symbol, that is, ac; the maximal suffix of the (sub)sequence acb is the (sub)sequence without its leftmost symbol, that is, cb, whereas a *suffix* of the acb comes out of acb by removing 0, 1, ..., |abc| symbols from the left of acb. The null sequence denoted as R is a suffix of any sequence and it holds that $|R|=0$.

The appearance count of subsequence **s**=ab is E(**s**)=E(ab)=2, and the normalized appearance count of **s** is equal to E(**s**) divided by the maximum number of (possibly overlapping) occurrences a subsequence of the same length could have, considering **a**'s length, that is, $E_n(\mathbf{s}) = E(\mathbf{s})/(|\mathbf{a}|-|\mathbf{s}|+1)$. The conditional probability of observing a symbol after a given subsequence is defined as the number of times that symbol has shown up right after the given subsequence divided by the total number of times that the subsequence has shown up at all, followed

by any symbol. Therefore, the conditional probability of observing the symbol b after the subsequence a will be denoted as $\overline{P}(b|a)$ and is equal to $\overline{P}$ (b|a)= E(ab)/E(a)=0.4.

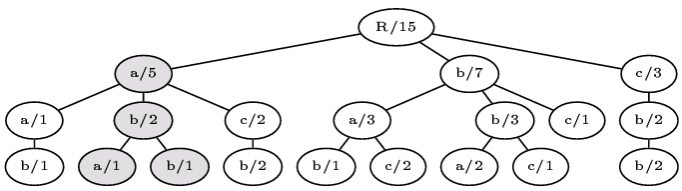
The generic procedure for deciding which subsequences will be inserted into the tree is specific to each family of Markov predictors and will be described in the next subsections. For purposes related to the clarity of presentation and comparison, we will build the respective tree of each family considering as input the sequence aabacbbabbacbbc. We will present the construction of each family's tree as simple as possible, omitting any optimizations, and we will assume that the input is given beforehand, although some predictors, that is, the online ones, do not demand the whole input to be known in advance.

The Prediction-by-Partial-Match Scheme

The prediction-by-partial-match scheme is based on the universal compression algorithm reported in Cleary and Witten (1984) and constructs a prediction model for an input sequence as follows. It assumes a predetermined maximal order, say, k, for the generated model. Then, for every possible subsequence of length of 1 up to k+1, if it has never been encountered before, we determine the node whose label is the maximal prefix of the considered subsequence. We create a new node under that node. The label of the new node is the length-1 suffix of the considered subsequence, and the new node's counter is initialized to the value of 1. If the considered subsequence has been encountered before, then the counter of the respective node is incremented by 1. Although this description implies that the whole input sequence is known in advance, the method works in an online fashion by exploiting a sliding window of size k+1 over the sequence as it grows symbol by symbol. The PPM predictor for the sample sequence aabacbbabbacbbc is depicted in Figure 1.

Upon completion of the construction phase, we can compute the probability of a symbol σ to appear after a context s by detecting the sequence sσ as a path in the tree emanating from the root, provided that $|s\sigma| \leq k$. The conditional probability of sσ is computed as the ratio of the node counter corresponding to sσ divided by the counter corresponding to σ. Therefore, having built the predictor of Figure 1, we can use it to carry out symbol prediction for a progressing sequence of events as follows: We determine the maximum context with length less than or equal to k that appears as a path in the tree, and compute the conditional probabilities of all symbols to appear after this context. For instance, adopting a most-probable prediction policy, the predicted symbol for the test context ab is a or b, and its conditional probability is 0.50 for either of them (see the gray-shaded nodes in Figure 1).

Figure 1. A PPM Markov predictor for the sequence aabacbbabbacbbc



The maximum context that the PPM predictor can exploit in carrying out predictions is k, though all intermediate contexts with length from 1 to k-1 can be used since they have already been stored into the tree. This model is also referred as the all-kth-order PPM model because it encodes a set of PPM predictors whose order ranges from 1 to k. The interleaving of various-length contexts does not mean that this scheme is a variable-length Markov predictor (although sometimes it is referred to as such) because the decision on the context length is made beforehand and not in a data-driven way.

Apart from this basic scheme, a number of variations have been proposed that attempt to reduce the size of the tree by pruning some of its paths or suffixes of some paths based on statistical information derived from the input data. They set lower bounds for the normalized appearance count and for the conditional probabilities of subsequences, and then prune any branch that does not exceed these bounds. Characteristic works adopting such an approach are reported in Chen and Zhang (2003), Nanopoulos et al. (2003), and Deshpande and Karypis (2004). Their basic motivation stems from the assumption that the pruned states add very little to the prediction capability of the original model, and thus they could be eliminated without sacrificing significantly its effectiveness. The validity of this assumption cannot be justified and, in any case, it strongly depends on the input data distribution. Apparently, these schemes are off line, making one or multiple passes over the input sequence in order to gather the required statistical information.

Application Fields

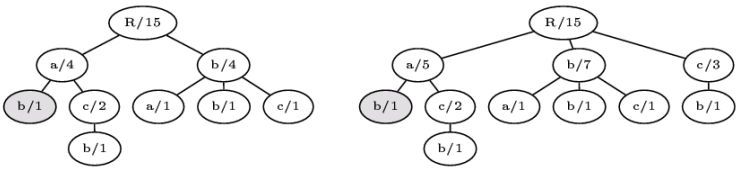
The PPM scheme was the first compression algorithm that was used in carrying out prediction in wired networks (Fan, Cao, Lin, & Jacobson, 1999; Palpanas & Mendelzon, 1999). Earlier, it had been exploited for the same task in databases (Curewitz, Krishnan, & Vitter, 1993). Although the currently implemented approaches, for example, in the Mozilla browser, implement link prefetching, the sophisticated procedure of the PPM could provide significant benefits.

The Lempel-Ziv-78 Scheme

The Lempel-Ziv-78 Markov predictor, LZ78 for short, is the second scheme whose virtues in carrying out predictions were investigated very early in the literature (Bhattacharya & Das, 2002; Krishnan & Vitter, 1998; Vitter & Krishnan, 1996). The algorithm LZ78 (Lempel & Ziv, 1978) arose from a need for finding a universal variable to the fixed-length coding method and constructs a prediction model for an input sequence as follows. It makes no assumptions about the maximal order for the generated model. Then, it parses the input sequence into a number of distinct subsequences, say, $\mathbf{s}_1$, $\mathbf{s}_2$, ..., $\mathbf{s}_x$, such that for all j, $1 \leq j \leq x$, the prefix of subsequence $\mathbf{s}_j$ (i.e., all but the last character of $\mathbf{s}_j$) is equal to some $\mathbf{s}_i$, for some $1 \leq i \leq j$. The discovered subsequences are inserted into a tree in a manner identical to that of the PPM scheme. In addition, the statistics regarding the number of appearances of each subsequence are stored into the nodes of the tree.

As the process of incremental parsing progresses, larger and larger subsequences are inserted into the tree, allowing the computation of conditional probabilities of increasingly larger

Figure 2. (Left) An LZ78 Markov predictor for the sequence aabacbbabbacbbc. (Right) An LZ78 predictor enhanced according to Bhattacharya and Das (2002)



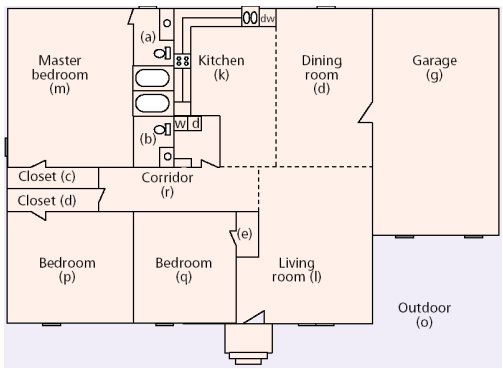
subsequences, thus exploiting larger contexts. The LZ78 predictor for the sample sequence aabacbbabbacbbc is depicted in the left part of Figure 2. The computation of conditional probabilities takes place in a manner completely analogous to that of PPM. However, LZ78 for this example is not able to produce a prediction for the test context ab (i.e., there is no subtree under the gray-shaded node).

Apparently, the LZ78 Markov predictor is an online scheme, it lacks administratively tuned parameters like lower bounds on appearance counts, and it is a characteristic paradigm of a variable-length Markov predictor. Although results do exist that prove its asymptotic optimality and its superiority over any fixed-length PPM predictor, in practice, various studies contradict this result because of the finite length of the input sequence. Nevertheless, the LZ78 predictor remains a very popular prediction method. The original LZ78 prediction scheme was enhanced in Bhattacharya and Das (2002), and Misra et al. (2004) in a way such that apart from a considered subsequence that is going to be inserted into the tree, all its suffixes are inserted as well (see right part of Figure 2).

Application Fields

Apart from the traditional use of the LZ78 algorithm in data-compression areas, recently it has found important application in problems related to location management in mobile networks. It has been proposed as a tool to reduce the communication overhead of the messages that are exchanged between the network and the roaming client (Misra et al., 2004; Roy, Das, & Misra, 2004). However, the applicability of the prediction algorithm is not confined to situations in which the alphabet is easily recognized; in the case of the wireless network, the alphabet consists of the cell IDs. We can have more general situations in which the application defines the symbols of the alphabet. The LZ78 algorithm has been used to track and predict the position of the inhabitants in smart-home applications (Das, Cook, Bhattacharya, Heierman, & Lin, 2002) in order to provide control over home devices. In this context, the house is modeled as a collection of nonoverlapping areas, which are later mapped into a symbolic map corresponding to the neighborhood information for each area. These notions are depicted in Figure 3 and Figure 4. Once we have done this mapping, the application of the prediction algorithm is straightforward.

Figure 3. The areas of a smart home

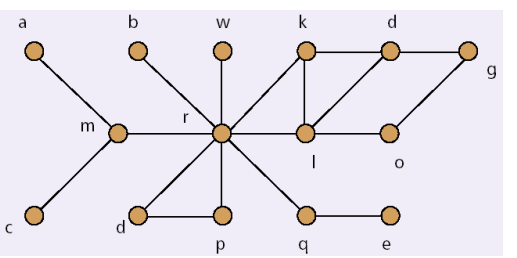


The Probabilistic Suffix Tree Scheme

The probabilistic suffix tree predictor, PST for short, was introduced in Ron, Singer, and Tishby (1996), and it presents some similarities to LZ78 and PPM. Although it specifies a maximum order for the contexts it will consider, it is actually a variable-length Markov predictor and constructs its tree for an input sequence as follows. The construction procedure uses five administratively set parameters: k, the maximum context length; a P_{min} minimum normalized appearance count for any subsequence in order to be considered for insertion into the tree; r, which is a simple measure of the difference between the prediction capability of the subsequence at hand and its direct father node; and γ_{min} and α, which together define the significance threshold for a conditional appearance of a symbol. Then, for every subsequence of length of 1 up to k, if it has never been encountered before, a new node is added to the tree, labeled by this subsequence in reverse symbol order provided that a set of three conditions hold. To exhibit the conditions, suppose that the subsequence at hand is abcd. Then, this subsequence will be inserted into the tree of the PST if

1. $E_n(abcd) \geq P_{min}$, and
2. there exists some symbol, say, x, for which the following relations hold:

Figure 4. The symbolic representation of the example smart home



a. $\frac{E(abcdx)}{E(abcd)} \geq (1+\alpha)\gamma_{\min}$, and

b. $\frac{\tilde{P}(x|abcd)}{\tilde{P}(x|abc)} \geq r \text{ or } \leq 1/r \equiv \frac{E(abc)}{E(abcd)} * \frac{E(abcdx)}{E(abcx)} \geq r \text{ or } \leq 1/r$

In addition, the node corresponding to the considered subsequence stores the (nonzero only) conditional probabilities of each symbol to appear after the subsequence. Obviously, the labels and statistics of each node of a PST differ from those of a PPM or LZ78 scheme. The PST predictor with the following set of parameters k=3, P_{min}=2/14, r=1.05, γ_{mim}=0.001, a=0 for the sample sequence aabacbbabbacbbc is depicted in Figure 5. Apparently, PST is a subset of the baseline PPM scheme when k is the same.

Upon completion of the construction phase, we can compute the probability of a symbol σ to appear after a context s by reversing s, and, starting from the root, can detect either the node whose label equals the reversed sequence or the deepest node whose label is a prefix of the reversed sequence. However, PST for this example is not able to produce a prediction for the test context ab (i.e., there is no subtree under the gray-shaded node).

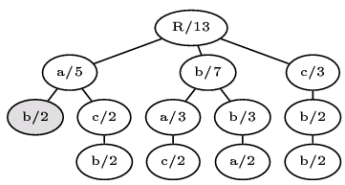
Application Fields

Apart from this basic scheme, a number of variations have been developed, the most important reported in Apostolico and Bejerano (2000), which provided improved algorithms, that is, linear algorithms for the procedures of learning the input sequence and making predictions. Other approaches adapt the technique to specialized contexts, such as computational biology (Bejerano & Yona, 2001; Largeron-Leteno, 2003). Currently, this scheme has not been applied in online problems such as location prediction, but it could be effectively employed in the request-prediction scenario under the assumption that the request pattern of the application does not change dramatically. It is relatively stable for large time intervals.

The Content-Tree Weighting Scheme

The context-tree weighting Markov predictor (Willems, Shtarkov, & Tjalkens, 1995), CTW for short, is based on a clever idea of combining exponentially many Markov chains of bounded order. The original proposition dealt with binary alphabets only, and its later

Figure 5. A PST Markov predictor for the sequence aabacbbabbacbbc



extensions for multialphabets (Volf, 2002) maintained this binary nature. For this reason, we will first describe the CTW Markov predictor for binary {0,1} alphabets and then give the most interesting and practical extension.

The CTW assumes a predetermined maximal order, say, k, for the generated model, and constructs a complete binary tree T of height k, that is, a binary tree in which every non-leaf node has two children and all leaves are at the same height k. An outgoing edge to the left-side children of T is labeled 0, and an outgoing edge to the right-side children of T is labeled 1. Each node s in T is associated with the sequence corresponding to the path from this node to the root. This sequence is also denoted as s. We can find to which input subsequence it corresponds by reversing it. The left and right children of node s are denoted as 0s and 1s, respectively.

Each node s maintains two counters, a_s and b_s, that count the number of 0s and 1s, respectively, that followed context s in the input sequence so far. Initially, all counters are set to 0. Then, we scan the input sequence by considering all subsequences of length k and for each subsequence, we update the counters of the nodes along the path defined by this subsequence. Additionally, each context (node) s maintains, apart from the pair (a_s, b_s), two probabilities, P_e^s and P_w^s. The former, P_e^s, is the Krichevsky-Trofimov estimator for a sequence to have exactly a_s 0s and b_s 1s, and it is computed as

$$\frac{\frac{1}{2}*\frac{3}{2}*\cdots\frac{2a_s-1}{2}*\frac{1}{2}*\frac{3}{2}*\cdots\frac{2b_s-1}{2}}{1*2*3*\cdots*(a_s+b_s)},$$

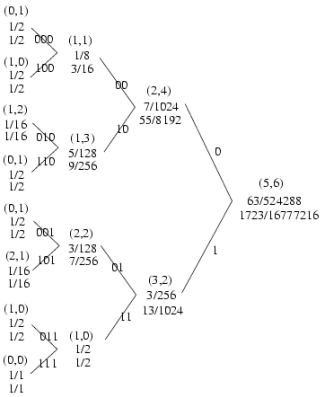
with $P_e^s(0,0)=1$, $P_e^s(1,0)=1/2$ and $P_e^s(0,1)=1/2$. The latter probability, P_w^s, is the weighted sum of some values of P_e, and it is computed with the following recursive formula:

$$P_w^s = \begin{cases} P_e^s & \text{for } |s| = k, \\ \frac{1}{2}P_e^s + \frac{1}{2}P_w^{0s}P_w^{1s} & \text{for } 0 \le |s| < k. \end{cases}$$

With P_e^R and P_w^R, we denote the Krichevsky-Trofimov estimate and the CTW estimate of the root, respectively. We can predict the next symbol with the aid of a CTW as follows. We make the working hypothesis that the next symbol is a 1, and we update the T accordingly, obtaining a new estimate for the root $P_e^{'R}$. Then, the ratio $P_w^{'R}/P_w^R$ is the conditional probability that the next symbol is a 1. If the next event is indeed a 1, we need not do any update to T; otherwise, we restore the previous values of the tree and perform the update that corresponds to appending a 0 to the input sequence. The CTW predictor for the sample binary sequence 010|11010100011 is depicted in Figure 6. The first three binary digits (at the left of |) are used to create a context for the sequence.

For the case of nonbinary alphabets, Volf (2002) proposed various extensions. We present the decomposed CTW, DeCTW for short, as the best compromise between method efficiency and simplicity. First, we assume that the symbols belong to an alphabet $\boldsymbol{\Sigma}$ with cardinality $|\boldsymbol{\Sigma}|$. We consider a full binary tree with $|\boldsymbol{\Sigma}|$ leaves. Each leaf is uniquely associated with a

Figure 6. A CTW Markov predictor for the binary sequence 010|11010100011

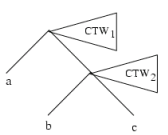


symbol in $|\Sigma|$. Each internal node v defines the binary problem of predicting whether the next symbol is a leaf on v's left subtree or a leaf on v's right subtree. Then, we attach a binary CTW predictor to each internal node. We project the training sequence over the relevant symbols (i.e., corresponding to the subtree rooted by v) and translate the symbols on v's left (respectively, right) subtree to 0s (respectively, 1s). After training, we predict the next symbol σ by assigning each symbol a probability that is the product of binary predictions along the path from the root of the binary tree to the leaf labeled by σ. A diagram of the DeCTW is depicted in Figure 7.

Application Fields

The inherent binary nature of the CTW prohibits its wider applicability. However, it has been successfully applied to some problems related to improving the performance of computer architecture. In particular, Federovsky, Feder, and Weiss (1998) described a direct application of the CTW method to the branch-prediction problem, which is the problem of assessing whether the program under execution will follow a branch or not. They looked at the program as a binary source that generates a binary sequence in which 1s correspond to taken branches and 0s correspond to not-taken branches. Furthermore, they model this binary symbol source using the CTW and perform branch prediction by blending the individual prediction models. Several efforts have been done toward alleviating the binary nature of the CTW and extending it for multialphabets. Of particular importance is the

Figure 7. A sketch of the DeCTW Markov predictor for the sequence aabacbbabbacbbc



predictor	sequence it models
CTW_1	aaxaxxxaxxaxxxx
CTW_2	bcbbbbcbbc

work by Sadakane, Okazaki, and Imai (2000), which provided a very simple and practical implementation along with a technique for combining the prediction strength of PPM and CTW (Okazaki, Sadakane, & Imai, 2002).

Comparison of Prediction Schemes

In the preceding section, we surveyed a number of Markov predictors. Implicitly or explicitly, they are all based on the short-memory principle, which, simply stated, says that the (empirical) probability distribution of the next symbol, given the preceding sequence, can be quite accurately approximated by observing no more than the last k symbols in that sequence.

Although this principle appears to be simple, the complications it introduces for the prediction algorithms are far from being simple. The algorithms are faced with the problems of selecting an appropriate value for k, which in general depends on the actual values of these most recent symbols. In absence of any other information, some methods fixed in advance the value of k (e.g., PPM, CTW). Such policies mainly suffer from the following drawback. If the value of k is too low and thus too general to capture all the dependencies between symbols, then the prediction efficiency of the respective model will not be satisfactory. On the other hand, if the value of k is too large, then the model will overfit the training sequence. Therefore, variable-length Markov predictors (e.g., LZ78, PST) are most appropriate from this point of view. This was the motivation for subsequent enhancements to PPM and CTW so as to consider unbounded-length contexts, for example, the PPM* algorithm (Cleary & Teahan, 1997).

On the other hand, variable-length predictors face the problem of which sequences and of what length should be considered. PST attempts to estimate the predictive capability of each subsequence in order to store it in the tree, which results in deploying many tunable parameters. LZ78 employs a prefix-based decorrelation process, which results in some recurrent structures being excluded from the tree, at least at the first stages. This characteristic is not very important for infinite-length sequences, but may incur severe performance penalty for short sequences or for sequences in which the patterns appear only a limited number of times; for instance, the pattern bba is missing in both variants of LZ78 of Figure 2.

Despite its superior prediction performance (for instance, see Effros, 2000), PPM is far less commonly applied than algorithms like LZ78. In practice, the LZ78 schemes are favored over PPM algorithms for their relative efficiencies in memory and computational complexity. While the LZ78 predictors can be implemented with O(n) memory and complexity, straightforward implementations of some PPM variants require worst case $O(n^2)$ memory and complexity to process a data sequence of length n (Cleary & Teahan, 1997). The high computational complexity of PPM algorithms remains an impediment for their more widespread use.

Finally, particular features encountered in each algorithm may make them less appealing for some applications. For instance, the CTW, due to its coupling with binary alphabets, is not the preferred choice for applications regarding multialphabets. Off-line schemes, for example, those in Deshpande and Karypis (2004) and Ron et al. (1996), are not appropriate when request prefetching must be performed by mobile clients.

Table 1. Qualitative comparison of discrete sequence-prediction models

Prediction Method		Overhead			Drawback
Family	**Markov Class**	**Training**	**Parameterization**	**Storage**	
LZ78	variable	online	moderate	moderate	misses patterns
PPM	fixed	online	moderate	large	fixed length high complexity
PST	variable	Off line	heavy	low	parameterization
CTW	fixed	online	moderate	large	binary nature

Table 1 summarizes the Markov-predictor families and their main limitations and advantages. While the considered features and metrics provide a general guideline for algorithm evaluation, the choice and performance of a specific model largely depends on the application.

A Suffix-Tree-Based Prediction Scheme

The suffix-tree-based prediction scheme, STP for short, is a new prediction algorithm not belonging to any of the aforementioned families, and it is described in Katsaros and Manolopoulos (2005). It works as follows. It finds the largest suffix of the input sequence s_1^n—let us call it ss_i^n—whose copy appears somewhere inside s_1^n. Then, it takes a suffix of ss_i^n (the length of this suffix is a parameter of the algorithm) and locates its appearances inside s_i^n. The symbols that appear after the appearances of it are the candidate predictions of the algorithm. The final outcome of the prediction algorithms is the symbol that appears the most times. In pseudocode language, the algorithms are presented in Figure 8.

To explain how the STP algorithm works, we present a simple example.

Example. Suppose that the sequence of symbols seen so far is the following:

s_1^{24}=abcdefgabcdklmabcdexabcd. The largest suffix of s_1^{24} that appears somewhere in s_1^{24} is ss_1^4=abcd. Let α=0.5. Then, sss_1^2= cd. The appearances of cd inside s_1^{24} are located at the positions 3, 10, 17, and 23. Therefore, the marked positions are 5, 12, 19, and 25. Obviously, the last one is not null since it contains the symbol we want to predict. In the general case, all marked positions will contain some valid symbol. Thus, the sequence of candidate predicted symbols is e, k, e. Since the symbol that appears most of the time in this sequence is e, the output of the STP algorithm, that is, the predicted symbol at this stage, is e.

The implementation of the algorithm requires an appropriate data structure to support its basic operations, which are the following: (a) the determination of the maximal suffix (at Step 1), and (b) substring matching (at Steps 1 and 2). These two operations can be optimally supported by a suffix tree. The suffix tree of a string $x_1, x_2, ..., x_n$ is a tree built from all suffixes of $x_1, x_2, ..., x_n$\$, where \$ is a special symbol not belonging to the alphabet. External nodes of a suffix tree contain information about the suffix positions in the original string

Figure 8. The STP algorithm

Algorithm *STP*
// Current sequence is $s_1^n = s_1, s_2, \ldots, s_n$.
// Predict the symbol after s_n.
begin
STEP 1.
Find the *largest* suffix of s_1^n, whose copy appears somewhere inside s_1^n.
Let this suffix be named ss_1^l. Its length is l and starts at
the position i in s_1^n, i.e.,
$ss_1^l = (s_{n-l+1}, s_{n-l+2}, \ldots, s_n) = (s_{n-i-l+1}, s_{n-i-l+2}, \ldots, s_{n-i})$.
STEP 2.
Take a suffix of ss_1^l of length k with $k = \lceil \alpha * l \rceil$, where α is a parameter.
Let this suffix be named sss_1^k, where $sss_1^k = (s_{n-k+1}, s_{n-k+2}, \ldots, s_n)$.
Suppose that ss_1^l appears m times inside s_1^n.
Each such occurence defines a *marker* and the m positions after
each market are called *marked positions*.
STEP 3.
The predicted symbol is the symbol that appears
the most times in the marked positions.
(In case of ties, the prediction consists of multiple symbols.)
end

and the substring itself that leads to that node (or a pair of indexes to the original string in order to keep the storage requirement linear in the string length). It is a well-known result that the suffix tree can be built in linear (optimal) time (in the string length), and can support substring finding in this string also in linear (optimal) time (in the length of the substring). Therefore, the substring searching operation of our algorithm can optimally be implemented. As for the maximal suffix determination operation, if we keep pointers to those external nodes that contain suffixes ending with the $ symbol (since one of them will be the longest suffix we are looking for), then we can very efficiently support this operation as well. From the above discussion, we conclude the following: (a) The STP algorithm is online, which means it needs no training or preprocessing of the historical data, (b) the storage overhead of the algorithm is low since it is implemented upon the suffix tree, and (c) the algorithm has only one tunable parameter α, which fine-tunes the algorithm's accuracy.

Further Research

This section presents a couple of directions that we feel would be significant to explore in future research. The first suggestion concerns the development of a new prediction model, and the second proposes to remove one of the assumptions that lead to the development of the current models.

The classical result about the duality between lossless compression (Feder & Merhav, 1994) and prediction implies that any universal lossless compression algorithm can be used to carry out prediction. Although quite a lot of theoretical lossless compression schemes do exist in the literature, only a few of them have been implemented for practical purposes.

This is due to the need for effectively combining prediction efficiency, computational complexity, and low implementation effort. These three dimensions limit the range of possible alternative, practical prediction models. Toward this direction, the Burrows-Wheeler (BW) lossless compression scheme offers significant opportunities (Effros, 2000) for combining the excellent prediction ratios of PPM and the low complexity of schemes based on LZ78. So far, no practical prediction scheme is based on the BW scheme, and a plethora of issues have yet to be considered to propose a practical model based on the BW method.

The cornerstone for building the Markov predictors described in this chapter is the "stationarity" assumption, which implied time-homogeneous transition probabilities. Under this assumption, the tree of each predictor grows node by node, increasing the respective node counters; that is, identical subsequences are aggregated (mapped) into the same node of the tree. If we remove the stationarity assumption, this technique is no longer appropriate. In the simplest case, for a mobile client whose roaming patterns change gradually, the predictors will tend to favor the old habits of the client and will adapt to the changing conditions at a very slow rate. Therefore, the assumption of non-time-homogeneous transition probabilities makes the current predictors inefficient and raises some design challenges for any new scheme that will be designed to address this assumption. As we mentioned, full aggregation is not helpful; partial (controlled) or no aggregation could be considered as well, but in any case, novel prediction algorithms should be designed. The technique reported in Ehrenfeucht and Mycielski (1992) could open some directions for research.

Conclusion

Discrete sequence prediction is an effective means to reduce access latencies and location uncertainty in networking applications. Due to the importance of the problem in various scientific fields, for example, machine learning, the Web, and bioinformatics, various methods have been reported in the literature. This chapter serves as a survey in this field, promoting the cross-discipline proliferation of ideas, although it by no means covers all proposed techniques. Important research issues have yet to be addressed, such as predictions for nonstationary sequences. We envision predictive model design as a fertile research area with both theoretical and practical solutions.

Acknowledgement

This research was supported by a Γ.Γ.Ε.Τ. grant in the context of the project Data Management in Mobile Ad Hoc Networks funded by the ΠΥΘΑΓΟΡΑΣ national research program.

REFERENCES

Apostolico, A., & Bejerano, G. (2000). Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *Journal of Computational Biology, 7*(3-4), 381-393.

Begleiter, R., El-Yaniv, R., & Yolan, G. (2004). On prediction using variable order Markov models. *Journal of Artificial Intelligence Research, 22*, 385-421.

Bejerano, G., & Yona, G. (2001). Variations on probabilistic suffix trees: Statistical modeling and prediction of protein families. *Bioinformatics, 17*(1), 23-43.

Bhattacharya, A., & Das, S. K. (2002). LeZi-update: An information-theoretic framework for personal mobility tracking in PCS networks. *ACM/Kluwer Wireless Networks, 8*(2-3), 121-135.

Buhlmann, P., & Wyner, A. J. (1999). Variable length Markov chains. *The Annals of Statistics, 27*(2), 480-513.

Chen, X., & Zhang, X. (2003). A popularity-based prediction model for Web prefetching. *IEEE Computer, 36*(3), 63-70.

Cleary, J. G., & Teahan, W. J. (1997). Unbounded length contexts for PPM. *The Computer Journal, 40*(2-3), 67-75.

Cleary, J. G., & Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications, 32*(4), 396-402.

Curewitz, K., Krishnan, P., & Vitter, J. S. (1993). Practical prefetching via data compression. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 257-266.

Das, S. K., Cook, D., Bhattacharya, A., Heierman, E., & Lin, T. Y. (2002). The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications Magazine, 9*(6), 77-84.

Deshpande, M., & Karypis, G. (2004). Selective Markov models for predicting Web page accesses. *ACM Transactions on Internet Technology, 4*(2), 163-184.

Effros, M. (2000). PPM performance with BWT complexity: A fast and effective data compression algorithm. *Proceedings of the IEEE, 88*(11), 1703-1712.

Ehrenfeucht, A., & Mycielski, J. (1992). A pseudorandom sequence: How random is it? *The American Mathematical Monthly, 99*(4), 373-375.

Fan, L., Cao, P., Lin, W., & Jacobson, Q. (1999). Web prefetching between low-bandwidth clients and proxies: Potential and performance. *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 178-187.

Feder, M., & Merhav, N. (1994). Relations between entropy and error probability. *IEEE Transactions on Information Theory, 40*(1), 259-266.

Feder, M., Merhav, N., & Gutman, M. (1992). Universal prediction of individual sequences. *IEEE Transactions on Information Theory, 38*(4), 1258-1270.

Federovsky, E., Feder, M., & Weiss, S. (1998). Branch prediction based on universal data compression algorithms. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 62-71.

Katsaros, D., & Manolopoulos, Y. (2004). Web caching in broadcast mobile wireless environments. *IEEE Internet Computing, 8*(3), 37-45.

Katsaros, D., & Manolopoulos, Y. (2005). A suffix tree based prediction scheme for pervasive computing environments. In *Lecture notes in computer science (LNCS)* (Vol. 3746, pp. 267-277). Springer-Verlag.

Krishnan, P., & Vitter, J. S. (1998). Optimal prediction for prefetching in the worst case. *SIAM Journal on Computing, 27*(6), 1617-1636.

Largeron-Leteno, C. (2003). Prediction suffix trees for supervised classification of sequences. *Pattern Recognition Letters, 24*, 3153-3164.

Merhav, N., & Feder, M. (1998). Universal prediction. *IEEE Transactions on Information Theory, 44*(6), 2124-2147.

Misra, A., Roy, A., & Das, S. K. (2004). An information-theoretic framework for optimal location tracking in multi-system 4G wireless networks. *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 286-297.

Nanopoulos, A., Katsaros, D., & Manolopoulos, Y. (2003). A data mining algorithm for generalized Web prefetching. *IEEE Transactions on Knowledge and Data Engineering, 15*(5), 1155-1169.

Okazaki, T., Sadakane, K., & Imai, H. (2002). Data compression method combining properties of PPM and CTW. In *Lecture notes in artificial intelligence (LNAI): Proceedings of the Conference on Progress in Discovery Science* (pp. 268-283).

Palpanas, T., & Mendelzon, A. (1999). Web prefetching using partial match prediction. *Proceedings of the 4th Web Caching Workshop (WCW)*.

Pitkow, J., & Pirolli, P. (1999). Mining longest repeating subsequences to predict World Wide Web surfing. *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, 139-150.

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning, 25*(2-3), 117-149.

Roy, A., Das, S. K., & Misra, A. (2004). Exploiting information theory for adaptive mobility and resource management in future wireless networks. *IEEE Wireless Communications Magazine, 11*(4), 59-65.

Sadakane, K., Okazaki, T., & Imai, H. (2000). Implementing the context tree weighting method for text compression. *Proceedings of the Data Compression Conference (DCC)*, 123-132.

Vakali, A. (2001). Proxy cache replacement algorithms: A history-based approach. *World Wide Web Journal, 4*(4), 277-297.

Vitter, J. S., & Krishnan, P. (1996). Optimal prefetching via data compression. *Journal of the ACM, 43*(5), 771-793.

Volf, P. (2002). *Weighting techniques in data compression: Theory and algorithms*. Unpublished doctoral dissertation, Technische Universiteit Eindhoven.

Willems, F. J., Shtarkov, Y. M., & Tjalkens, T. J. (1995). The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory, 41*(3), 653-664.

Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory, 24*(5), 530-536.

Section III

Web Information Integration and Applications