Efficient Methods on Predictions for Similarity Search over Stream Time Series

Xiang Lian Lei Chen
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
\{xlian, leichen\}@cs.ust.hk

Abstract

Due to the wide usage of stream time series, an efficient and effective similarity search over stream data becomes essential for many applications. Although many approaches have been proposed for searching through archived data, because of the unique characteristics of the stream, for example, data are frequently updated, traditional methods may not work for the stream time series. Especially, for the cases where the arrival of data is often delayed for various reasons, for example, the communication congestion or batch processing and so on, queries on such incomplete time series or even future time series may result in inaccuracy. Therefore, in this paper we propose two approaches, polynomial and probabilistic, to predict the unknown values that have not arrived at the system. We also present efficient indexes, that is, a multidimensional hash index and B+-tree, to facilitate the prediction and similarity search on future time series, respectively. Extensive experiments demonstrate the efficiency and effectiveness of our methods in terms of I/O, prediction and query accuracy.

1. Introduction

Recently, stream time series data management has become a hot research topic due to its wide application usages, such as Internet traffic analysis [7], sensor network monitoring [34], moving object search [10], financial data analysis [28], and the like. All these applications require continuously monitoring stream time series data. Compared to traditional archived time series data, stream time series data have their own characteristics, that is, data are frequently updated. Therefore, previous methods for similarity search over archived time series data may not work in this scenario. For example, in sensor networks, the system continuously receives data from sensors, which are often delayed due to various reasons, such as communication congestion in the network or batch processing [2], and so on. In this case, for similarity queries over the time when data are not being updated, previous approaches have only performed searches on incomplete time series (on previously received time series data). Thus, the answer is inaccurate due to the missing data. Furthermore, some other applications may even request a similarity search on the data some time in the future. For example, in the stock market, we might want to know which stocks follow a certain pattern over a few days, so that we can take appropriate action to either make a profit or reduce losses. Obviously, in such a case, we do not have any knowledge of the exact future subsequences, so queries can not be answered without guesswork.

Motivated by this, in this paper, we focus on tackling the problem of predicting values that have not arrived in the stream time series and building efficient indexes for both a prediction and similarity search on these unknown series.

Previous work on predicting future data mainly use fuzzy methods [19, 23] or data mining techniques [14, 27, 29] to extract features from the training dataset and perform the prediction tasks on real time series. However, in order to achieve accurate results that are insensitive to the evolving data, these methods usually require training the predictors on the actual data at high cost. Therefore, although these approaches can lead to good offline prediction accuracy given appropriate training, they are generally not suitable for the online stream environment, which is in favor of low prediction and training costs. Moreover, previous methods often design complicated models that are specific to some concrete applications, for example, prediction in binary time series [13]. However we are seeking general solutions that are straightforward yet effective.

The main contributions of our paper are summarized as follows.

- We present the polynomial approach which predicts future values based on the approximated curve of the most recent values.
- We propose another prediction method, the probabilistic approach, to predict future values according to aggregates of all the data in history.
- Furthermore, our probabilistic approach is adaptive to the possible change of data because it can efficiently feed the prediction error back.
- Last but not least, we present indexes for the probabilistic approach which efficiently facilitate prediction and similarity queries in the stream environment.

The rest of the paper is organized as follows. Section 2 reviews the related work on the similarity search on the archived time series and prediction techniques.
Section 3 formally defines our problem and illustrates the general framework. Section 4 demonstrates our polynomial and probabilistic approaches. Section 5 discusses indexes for the probabilistic method. Section 6 evaluates the prediction and query accuracy as well as efficiency with extensive experiments and Section 7 concludes this paper with directions for future work.

2. Related Work

Section 2.1 reviews the traditional similarity search in time-series databases, as well as various dimensionality reduction techniques. Section 2.2 briefly presents the previous work on value predictions.

2.1. Similarity Search in Time-series Databases

Similarity search over time series data has been studied for many years. Most of the proposed methods have focused on searching over historical data, which is, given a query sequence $Q = \{q_1, q_2, \ldots, q_L\}$ of length $L$ and similarity threshold $\epsilon$, the similarity query finds all the (sub)sequences $S = \{s_1, s_2, \ldots, s_k\}$ with length $L$ in the database that are similar to $Q$, that is, satisfying $\text{dist}(Q, S) \leq \epsilon$, where the distance function $\text{dist}(Q, S)$ between $Q$ and $S$ can be $L_p$-norms [1, 12], DTW [3, 18, 20, 32], LCSS [5, 11, 26], EDR [10], or ERP [8], and so on. For simplicity, throughout this paper, we use the Euclidean distance (L_2) between $Q$ and $S$ as our metric, which is defined as follows:

$$\text{dist}(Q, S) = \sqrt{\sum_{i=1}^{L} (q_i - s_i)^2} \quad (1)$$

In order to efficiently perform the similarity search, Agrawal et al. [1] and Faloutsos et al. [12] transform the (sub)sequences of length $L$ to lower $s$-dimensional points ($\ll L$) with a dimensionality reduction technique, Discrete Fourier Transform (DFT) [1], and then insert them into an R-tree [4]. Other reduction techniques include SVD [16], DWT [24], PAA [31], APCA [17], Chebyshev Polynomial [9], and so on. All these techniques follow the lower bounding lemma, that is, the distance between any two converted $s$-dimensional points is never greater than that between the original points in $L$-dimensional space.

In summary, previous work on similarity search mainly based on complete archived time series. Our work, however, focuses on queries over the future time series data.

2.2. Prediction Techniques

Related work on predicting values involves techniques that apply fuzzy rules [19, 23] and data mining approaches [14, 27, 29]. The problem of predicting unknown values is defined as follows. Assume we know $H$ consecutive values, $x_1, x_2, \ldots, x_H$, and want to predict the next $\Delta$ future values $x_{H+1}$, $x_{H+2}$, $\ldots, x_{H+\Delta}$. The traditional fuzzy predictor predicts values directly on the raw time series, where the fuzzy rules are in the form "if $x_1 = \text{value}_1, x_2 = \text{value}_2, \ldots, x_H = \text{value}_H$, then $x_k = \text{value}_k$, $k \in [H+1, H+\Delta]$". However, this approach can guarantee the prediction accuracy only if the time series' statistics are stationary. Otherwise, with a dynamic series it might fail to give accurate results. Kim et al. [19] proposed a new predictor which is based on rules with differences of consecutive values, that is, "if $x_1, x_2 = \text{value}_1, x_2 - x_1 = \text{value}_2$, $\ldots, x_{H+1} - x_{H} = \text{value}_{H+1}$, then $x_{H+1} = \text{value}_{H+1}$, where $k \in [H+1, H+\Delta]$". Policker et al. [23] present methods to classify the time series using the fuzzy clustering algorithm, calculating the fuzzy membership of each possible result, and producing the estimation of future elements.

Gestel et al. [14] make use of Least Squares Support Vector Machines (LS-SVM) to infer non-linear models and predict the financial time series. Wang et al. [29] propose wavelet packet MLP (WP-MLP) for prediction, based on the work of the wavelet multilayer perceptron (W-MLP) by Zhang et al. [33]. Vilalta et al. [27] present the target (true) events in the database of event sequences by mining the frequent event sets that occur associated with the target events and building up a rule-based model for prediction.

As we have discussed, these approaches, for example, clustering or training the neural network, incur very high update cost for either mining fuzzy rules or training parameters in different models. Therefore, they are not applicable to efficient online processing in the stream environment, which is in favor of low prediction and training cost.

3. Problem Definition and Framework

In this section, we formally define the similarity search in the future time series problem and present our two goals. Figure 3.1 illustrates the general framework which mainly focuses on two tasks, that is, (i) predicting future values for each time series, and (ii) answering similarity queries on subsequences in the future. Assume we have $n$ stream time series $T_1, T_2, \ldots, T_m$ in the stream environment, each containing $m$ ordered values at the current timestamp ($m$-1), that is, $T_1 = \{t_0, t_1, \ldots, t_{(m-1)}\}$ where $t_j$ is the value at timestamp $j$ in $T_1$. Due to the delay of the data, for example, the batch processing [2], these $n$ stream time series would receive data only after $\Delta$ timestamps. In other words, for each time series $T$, the future values $t_{(m+1)}, t_{(m+2)}, \ldots, t_{(m+\Delta-1)}$ corresponding to timestamps $m$,$(m+1),$,$\ldots,$ and $(m+\Delta-1)$, respectively, arrive in a batch manner at the same timestamp $(m+\Delta)$. Therefore, during the period from timestamp $m$ to $(m+\Delta-1)$, called the blocked period, the system knows nothing about $\Delta$ future values in each series. In order to answer similarity queries on future subsequences, we have to use a predictor to predict these unknown values based on the most recent $H$ values and extract subsequences from them. After the actual data arrive at the system, the predictor is trained correspondingly.

The first goal of our work is to efficiently predict the $n \times \Delta$ future values for $n$ time series, so that the prediction error is as low as possible. In particular, we
define the prediction error as the squared Euclidean distance (defined in Formula (1)) between the predicted and actual values. That is, for time series \( T_i \), the prediction error \( \text{Err}_{\text{pred}}(T_i) \) is measured by:

\[
\text{Err}_{\text{pred}}(T_i) = \sum_{j=m}^{m+\Delta t-1} (t_{ij} - t_{ij}')^2
\]

where \( t_{ij}' \) corresponds to the predicted value of \( t_{ij} \).

![Figure 3.1: The Framework of Our Solution](image)

We denote as \( T[j : k] \) the subsequence \( \{t_{ij}, t_{i(j+1)}, \ldots, t_{ik}\} \) of \( T_i \), where \( j \leq k \). Those subsequences that contain at least one future value are called future subsequences. Specifically, \( T[j : k] \) is a future subsequence if and only if \( m \leq k \leq (m+\Delta t-1) \) holds. Given a query sequence \( Q \) of length \( L \), the similarity search in the future time series is to retrieve all future subsequences \( T[k-L+1 : k] \) with length \( L \) from \( T_i \), such that \( \text{dist}(T[k-L+1 : k], Q) \leq \epsilon \), where \( m \leq k \leq (m+\Delta t-1) \). Therefore, our second goal is to extract all future subsequences from \( n \) time series with predicted values and then efficiently answer similarity queries on them, so that the query accuracy is as high as possible. In particular, the query accuracy is measured by the recall ratio of the query result formulated as follows:

\[
\text{Recall ratio} = \frac{\text{recall num}}{\text{act num}}
\]

where \( \text{recall num} \) is the number of candidates in the query result that indeed match with the query sequence \( Q \), and \( \text{act num} \) is the actual number of future subsequences that are in \( \epsilon \)-match with \( Q \). Table 3.1 summarizes the commonly-used symbols in this paper.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_i )</td>
<td>the ( i )-th time series</td>
</tr>
<tr>
<td>( n )</td>
<td>the number of time series</td>
</tr>
<tr>
<td>( m )</td>
<td>the number of data in series at present</td>
</tr>
<tr>
<td>( L )</td>
<td>the length of the query ( Q )</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>similarity threshold</td>
</tr>
<tr>
<td>( H )</td>
<td>the number of historical data used to predict</td>
</tr>
<tr>
<td>( \Delta t )</td>
<td>the number of predicted values</td>
</tr>
<tr>
<td>( K )</td>
<td>the cardinality of the alphabet table</td>
</tr>
<tr>
<td>( s_i )</td>
<td>the symbol in the alphabet table</td>
</tr>
<tr>
<td>( l )</td>
<td>the length of each segment in PAA</td>
</tr>
<tr>
<td>( h )</td>
<td>the height of the trie (( h = H/l ))</td>
</tr>
</tbody>
</table>

**Table 3.1: Symbols and Their Descriptions**

4. Two Prediction Approaches

In this section, we present two approaches, polynomial and probabilistic, to predict the future values of the stream time series.

4.1. Polynomial Prediction

As mentioned before, we are interested in prediction techniques that are suitable for the stream environment, which is in favor of low prediction and training cost. One approach that satisfies this requirement is to (i) use some commonly-used curves to approximate the historical data, and (ii) predict future values according to curves.

Here, we choose the polynomial curve, since it is simple and efficient for online processing. In particular, among the family of polynomial curves, we focus on two types, linear and quadratic [25]. Figure 4.1 illustrates an example of predicting the time series with linear and quadratic prediction methods. Following the convention [6, 15], we assume that all the values in time series \( T_i \) have a domain [\( \min, \max \)].

![Figure 4.1: Example of Polynomial Prediction](image)

Assume we use \( H \) values \( x_1, x_2, \ldots, x_H \) to predict \( \Delta t \) consecutive values \( x_{H+1}, x_{H+2}, \ldots, x_{H+\Delta t} \) in the future. Without loss of generality, let \( x_1 \) be the value at timestamp 1, \( x_2 \) at timestamp 2, and so on. We first consider the linear prediction, which assumes all the \( (H+\Delta t) \) values can be approximated by a single line in the form \( x = at + b \), where \( t \) is the timestamp, \( x \) is the estimated value, and parameters \( a \) and \( b \) characterize these \( (H+\Delta t) \) data. Therefore, the predicted \( \Delta t \) values with a linear predictor are \((H+1)at+b, (H+2)at+b, \ldots, (H+\Delta t)at+b\), corresponding to \( x_{H+1}, x_{H+2}, \ldots, x_{H+\Delta t} \), respectively.

We measure the approximation error of the linear curve on historical data by the squared Euclidean distance between the actual series and the estimated one. That is,

\[
\text{Error}_{\text{linear appr}} = \sum_{i=1}^{H} (a(i+b)-x_i)^2.
\]

In order to obtain a good approximation, we aim at finding appropriate coefficients \( a \) and \( b \), such that the error \( \text{Error}_{\text{linear appr}} \) is minimized. This can be achieved by the differential of \( \text{Error}_{\text{linear appr}} \). Specifically, \( a \) and \( b \) must satisfy two conditions: (i) \( \partial \text{Error}_{\text{linear appr}} / \partial a = 0 \), and (ii) \( \partial \text{Error}_{\text{linear appr}} / \partial b = 0 \).

Similarly, for the quadratic prediction, we approximate values by a quadratic curve in the form \( x = at^2 + bt + c \), where \( a, b \) and \( c \) are parameters that characterize the data. The approximation error of a quadratic curve is measured by:

\[
\text{Error}_{\text{quad appr}} = \sum_{i=1}^{H} (a(i^2+b)+c-x_i)^2.
\]

In order to minimize the error, three conditions must
hold: (i) \( \partial \text{Error}_{\text{quad}, \text{app}} / \partial a = 0 \), (ii) \( \partial \text{Error}_{\text{quad}, \text{app}} / \partial b = 0 \), and (iii) \( \partial \text{Error}_{\text{quad}, \text{app}} / \partial c = 0 \). In the cases of both \textit{linear} and \textit{quadratic} predictions, coefficients \((a, b, \text{and } c)\) can be obtained by solving equations with \textit{Cramé}'s rule.

As in Figure 4.1, when the approximating curve (either \textit{linear} or \textit{quadratic}) intersects with the lower/upper bound \textit{min/max}, the predicted values after this timestamp would become meaningless. Furthermore, if the number of predicted values that are meaningful is not greater than \( \Delta t \), curves of higher orders have to be used. Nevertheless, the computational cost is high and moreover the prediction accuracy may not even be as good as low-order curves. Therefore, for simplicity, when the predicted value is below \textit{min} (above \textit{max}), we just treat the value as \textit{min} (\textit{max}).

Another important issue is how to choose the value of \( H \). We build up histograms for angles of any two consecutive values on historical data, and find that, with high variance of the histogram, indicating the irregularity of the underlying data, a smaller value of \( H \) can achieve higher accuracy. Since the cost of either creating or maintaining the histogram is linear, it is especially suitable for tuning \( H \) during the online stream processing. In this way, our method can be adaptive to the change of data.

### 4.2. Probabilistic Prediction

Our second prediction approach is based on the observation that, those subsequences that appear frequent in history have a higher probability of occurring again in the future. Motivated by this, we utilize statistics on the entire historical data, rather than a few most recent in the \textit{polynomial} solution, to predict future values. Specifically, we propose our \textit{probabilistic} approach, which extracts the symbolic representation of historical subsequences as well as their aggregate information, to predict the future symbols with probabilities based on aggregates that summarize the entire historical data, and finally output \textit{future subsequences} for similarity search.

Without loss of generality, we consider the prediction problem on a single time series \( T \), which can be easily extended to the case of \( n (>1) \) time series. Details of the approach are as follows. Assume at the current timestamp, we know the most recent \( H \) historical data and have to predict the future \( \Delta t \) values. In order to store the statistics in history, we maintain a structure, called the \textit{aggregate trie}. Specifically, for each historical subsequence of length \( H \), we predict the future \( \Delta t \) values in 4 steps:

- **Divide it into** \( h \) disjoint segments with identical length \( l \), that is, \( H = h \cdot l \).
- **Obtain the mean value** \( \text{avg}_i \) for each segment \( i \), where \( 1 \leq i \leq h \).
- **Convert each** \( \text{avg}_i \) into a symbol \( s_i \), that is, transforming the subsequence to its symbolic representation \( s_1 s_2 \ldots s_h \), and
- **Insert the string into** the aggregate trie with height \( h \).

The first three steps transform the time series to its symbolic representation proposed by Lin et al. [21]. Figure 4.2 illustrates an example of this transformation.

Assume the value domain of a time series \( T \) is \([-2, 2]\). We partition it into three smaller ranges, say \([0.5, 2]\), \([-0.3, 0.5] \) and \([-2, -0.3] \), which correspond to three symbols \( a, b, \) and \( c \), respectively. As a second step, we divide the time series \( T' \) of length \( H \) into 7 segments of equal length, take the average value \( \text{avg}_i \) within each segment \( i \), and finally convert each \( \text{avg}_i \) into a unique symbol. For example, the mean value \( \text{avg}_2 \) in the first segment falls into the range \([0.5, 2]\), so we map it to the symbol “\( a \)”. After the discretization, \( T' \) is represented by a string consisting of ordered symbols (also called \textit{SAX} representation). Specifically, the discrete version of \( T' \) is the string “abcbbaa”. The main advantage of discretizing the time series is its space efficiency. That is, if there are totally \( K \) symbols in the alphabet table, then each sequence requires only \( h \cdot \lfloor \log_2(K) \rfloor \) bits at most, where \( h = H / l \).

The fourth step inserts all the time series strings into a trie. However, distinguished from the ordinary trie, each node entry in our aggregate trie contains a triple \(<\text{freq}, \text{hit}, \text{miss}>\), where \text{freq} is the frequency that a string (from the root to leaf) appears in the time series, \text{hit} is the times that our prediction succeeds and \text{miss} is that it fails. Intuitively, if the frequency \text{freq} of a particular string is high, then it will have a higher probability of appearing in the future. On the other hand, if our prediction of a certain string fails quite often, that is, its aggregate \text{miss} is large, then we have to lower the chance of choosing this string as the prediction result. In other words, our \textit{prediction error} can be fed back by such aggregates.

Figure 4.3 illustrates an example of converting subsequences into strings and inserting them into a trie. Assume \( h = 4, l = 2 \) and \( \Delta t = 6 \) (i.e., \( H = 8 \)). Initially, the aggregates in each node of the trie are in the form \(<0, 0, 0>\). Let the current timestamp be 7. The historical subsequence \( T[0:7] \) of length 8 is first transformed to a string “aabbaa”. Then, we insert it into the trie through the path “aabba”. Meanwhile, increase by 1 the frequency \text{freq} of each entry on the path in the form \(<\text{freq}, \text{hit}, \text{miss}>\), that is, updating \(<0, 0, 0>\) to \(<1, 0, 0>\) in the example.

Next, we illustrate the process of predictions using the example of Figure 4.3. In particular, we want to predict symbols \( X_1, X_2, \ldots, \) and \( X_6 \) based on the trie,
Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM’06)

0-7695-2590-3/06 $20.00 © 2006 IEEE

Figure 4.3: Probabilistic Prediction with Aggregate Trie corresponding to subsequences T[7:8], T[8:9], ..., and T[12:13], respectively. For simplicity, we only consider predicting the symbol $X_i$. Since the symbolic representation of $T[1:8]$ is "aaccX", we traverse the trie from the root to entry "a" on level 4, and then "a" and "c" on level 3 and 2, respectively. After we have reached "c" on level 2, we should decide which symbol is most likely to be $X_i$, that is, the procedure of the prediction.

One intuitive idea is to select the next symbol with the highest frequency. For example, if the frequency of both symbols "a" and "c" is 0 while that of "b" 2, then the string "aacb" has higher probability than "aaca" or "aacc" to appear in future. So we can choose b as $X_i$. Although this intuitive idea can predict the future based on historical data, it can not quickly adapt to the change of data. In this paper, we predict future values based on both frequency and hit ratio. Since we also store hit and miss information in each entry of the trie, our approach takes into account the feed-back of the prediction error in a probabilistic way. In particular, for each symbol $s_i$ with <$freq_{s_i}$, hit_{s_i}, miss_{s_i}>$, the probability $Prob_{s_i}$ of selecting $s_i$ as $X_i$ is proportional to $freq_{s_i} \cdot hit_{s_i} / (hit_{s_i} + miss_{s_i})$. Moreover, in the string "aaccX", suffixes "ac" and "c" of "aacc" can also be used to predict $X_i$, but in a less accurate way due to fewer known symbols. Therefore, we can assign decreasing weights to "aacc", "ac" and "c" during predictions. In the sequel, however, we only consider predicting future symbols with the longest string, for example, "aacc", for simplicity. Note that, in some situations, for example, the freq aggregates of all the possible symbols of $X_i$ is zero, we handle such exceptions by simply setting $X_i$ to its previous symbol in the string, that is, "c" in string "aaccX". Let $Prob_b$ be the probability of predicting $X_i$. Similarly, other symbols, such as $X_2$, $X_3$, ..., and $X_n$, are predicted with probabilities $Prob_2$, $Prob_3$, ..., and $Prob_n$, respectively.

After all the unknown symbols have been predicted, future subsequences of length $L$ are extracted without any difficulty. During the blocked period, whenever a query sequence $Q$ of length $L$ arrives, we can (i) perform the similarity search on these subsequences (in symbolic representation), and (ii) output candidates whose lower bound distances from $Q$ are not greater than the similarity threshold $e$. Issues of indexing future subsequences are discussed in the next section.

Finally, at timestamp 14, data $T[8]$, $T[9]$, ..., and $T[13]$ will arrive in a batch. At this point, the actual symbols of $X_1$, $X_2$, ..., and $X_6$ come out. As an example, for symbol $X_1$, if the prediction of $X_i$ is $b$ whereas the actual one is $a$ (wrong prediction), then we feed the error back by increasing miss in all entries <$freq$, hit, miss$> on the path of "aacb" by $Prob_b$, and $freq$ on that of "aaca" by $H_1$. Otherwise, if the actual symbol of $X_i$ is indeed $b$ (correct prediction), then in all entries on path "aacb", we increase $freq$ by 1 and hit by $Prob_b$. For other symbols, the procedure of updating aggregates in the trie is similar. Note that, however, for "aX_1X_2" of $T[3:10]$, when we update either a hit or miss aggregate, we always increase its value by $Prob_b$ of $x_3$, since the prediction of $X_1$ is based on the predicted result of $X_1$.

Figure 4.4: Choosing Appropriate Values of $H$

Selection of $H$. Figure 4.4 illustrates an example of prediction with two different values of $H$, $H_1$ and $H_2$. If we build a trie with the large value $H_1$ of an $H$, pattern $A$ may appear very frequently, thus having more chance to predict a sharp curve in the future timestamps. On the other hand, for a smaller value $H_2$ of $H$, the trie we construct will contain pattern $B$ with high frequency, so future values are more likely to be smooth. Now the problem arises how to choose the appropriate value of $H$. Our solution is to ensure the accuracy of user-specified queries. That is, if the user specifies the query $Q$ of fixed length $L$, the value of $H$ should be about the same value as $L$. In the previous example, the length $L$ of $Q$ is close to $H_1$, that is, $Q$ has higher probability to capture a sharp curve. Therefore, we choose $H_1$ to build the trie. In the case of various query lengths, we suggest using different values of $H$, for example, $H_{min}$, $2H_{min}$, ..., and $2^{L-1}H_{min}$ that are exponentially increasing, where $H_{min}$ is the minimum possible query length, and $2^{L-1}H_{min}$ the maximum. When a query of length $L$ arrives, we select the trie with $2^{L-1}H_{min}$ such that $2^{L-1}H_{min} \leq (L+\Delta L) \leq 2^LH_{min}$. In this way, our approach can achieve good query accuracy which is adaptive to the query length $L$.

5. Indexes for the Probabilistic Approach

In previous sections, we only consider prediction and similarity search on a single stream time series. In such scenario, since there is only one trie for a single time series and the number $\Delta t$ of predicted symbols is relatively small, in most cases, both prediction and similarity search can be processed in memory. However, in the scenario of $n$ stream time series, for example, thousands of time series, the problem becomes more complex. For one thing, it is very unlikely that the memory can retain $n$ tries (built from $n$ series, respectively) and $n\Delta t$ predicted symbols without any
overflow. Once we decide to flush some tries or predicted symbols back to the disk, the I/O cost becomes the bottleneck of the online processing, especially when tries are of a great height. Another thing, without indexes, the sequential scan might be the only way to perform the similarity search on n-d predicted subsequences, but with very high cost.

Motivated by this, we aim to build up efficient indexes to reduce the I/O cost. Section 5.1 discusses predictions, Section 5.2 illustrates the similarity search, and Section 5.3 presents the update methods to aggregate (i.e., error feedback).

5.1. The Alternative Index for Trie

As we have mentioned, it is not feasible to maintain n aggregate tries separately with great height h. Furthermore, we do not even need to materialize an actual trie. This is based on the observation that, each path of the trie can be uniquely identified by a number of ordered integers (key). As an example, given the descending path "acb" of a trie, instead of traversing the trie from the root to leaf, we can alternatively retrieve entries with a multidimensional key corresponding to the path, for example, a unique key <1, 3, 2> of "acb", assuming a, b and c are mapped to integers 1, 2 and 3, respectively.

Based on the above idea, we construct a single h-dimensional hash index HI for all the n time series. Here, we choose the hash index because of its good retrieval efficiency. That is, the cost of either searching or inserting one data in the hash structure is about O(1). The only difference of our index from the ordinary hash, or inserting one data in the hash structure is about (1).

The second index we propose is a B+-tree FI, which temporarily stores all the predicted future subsequences and facilitates the similarity search. In particular, the search key in FI is computed by converting h symbols of a predicted string into a single value, similar to the calculation of Hash(PA) in HI mentioned before. Each entry of the leaf node is in the form <Hash(PA), tid, freq, hit, miss>, where Hash(PA) is a hashing function converting the path PA of length h in the trie into a single value, tid is the id of time series, and the last three attributes are aggregates the same as those stored in the trie. Note that here the string "s1s2s3s4s5s6" on the path PA can be mapped to a value Hash(PA) by concatenating all the bits of h integers, i.e., si1, i2, ..., and ih, from h symbols s1, s2, ..., sih, respectively. As an example, assume there are 3 symbols s1, s2, and s3 in total and the height h of a trie is 4. The string "s1s2s3s4" can be converted into a value "01,11,10,01" in bit representation and Hash("s1s2s3s4") is thus 121 in decimal representation. Note that given a hashed value Hash(PA), we can also reversely obtain its original string "s1s2s3s4". Entries in the bucket are sorted in the order of tid and Hash(PA) for the convenience of prediction.

5.2. The Index for Similarity Search

The second index we propose is a B+-tree FI, which temporarily stores all the predicted future subsequences and facilitates the similarity search. In particular, the search key in FI is computed by converting h symbols of a predicted string into a single value, similar to the calculation of Hash(PA) in HI mentioned before. Each entry of the leaf node is in the form <Hash(PA), tid, offset, ptr>, where Hash(PA) is the key, tid the id of time series, offset the end offset of the predicted subsequence, ptr a pointer pointing to the corresponding entry in HI.

During the blocked period, all the similarity search queries are performed on FI. In particular, given a query Q of length L, our goal is to find candidate subsequences in future such that the lower bound distance from Q to candidates is within ε. The basic idea is as follows. Assume the height of the tree FI is h', and the segment mean representation of Q is SM(Q) (in reverse order). Starting from the root of FI, we obtain all the possible \(|L(h')|\) symbols that have the squared lower bound distance LB_dist from the first \(|L / (h')|\) segment means of Q within ε, and transform them to value intervals. Descend from a branch only if the value intervals intersect with its key range. For each child node, we repeat the procedure, but with a smaller similarity threshold \(\sqrt{\varepsilon^2 - LB_{dist}}\). We output those candidates on the leaf nodes with the similarity threshold greater than or equal to zero. Since we use SM(Q) as the query during the search, false alarms may still exist. Therefore, as a second step, we use the actual values in Q to further refine the candidate set.
5.3. Combining All Together

By combining all the indexes together, we illustrate the entire process of the similarity search in future time series, in an example of Figure 5.1. Assume the alphabet table is \{a, b, c\} and the height h of the trie is 3. Let a be "01" in bit representation, b "10", and c "11" respectively. At the current timestamp, we consider predicting the future string "acX" for time series \(T_t\). First (step 1(a)), we convert 3-dimensional vector <a, c, 1> into a single value "011" in bit by extracting the first bit from a ("01") and c ("11"), respectively, and the last bit "1" of \(T_1\). By searching "011" in \(H_{II}\), we find three entries <Hash\(\"aca\"\), 1, freq_a, hit_a, miss_a>, <Hash\(\"acb\"\), 1, freq_b, hit_b, miss_b> and <Hash\(\"ace\"\), 1, freq_c, hit_c, miss_c> in a bucket, corresponding to strings "aca", "acb", and "ace", respectively. As discussed in the previous section, we compute the probability of selecting each entry as our prediction result and choose the one proportional to its probability. Without loss of generality, let the chosen symbol be b with probability Prob_b. Next (step 1(b)), we construct a B-tree FI for indexing future subsequences. In particular, we insert the entry <Hash\(\"bca\"\), 1, offset, ptr> into FI with the key Hash\(\"bca\"\) and pointer ptr pointing to <Hash\(\"acb\"\), 1, freq_b, hit_b, miss_b> in \(H_{II}\), where offset is the end offset of the future subsequence. Note that, here we calculate the key in the reverse order of the original string "acb" for the convenience of similarity search. Predictions of other future subsequences are similar.

During the blocked period, we always perform the similarity search on index FI (step 2). Then, after \(\Delta\) timestamps, the actual data arrive at the system. We batch scan all the leaf nodes in FI (step 3(a)). For each entry we encounter, for example, <Hash\(\"bca\"\), 1, offset, ptr>, we obtain the true value of \(X_i\) in the subsequence with end offset offset in time series \(T_t\). If \(X_i\) is indeed b (as we predicted), access the entry <Hash\(\"acb\"\), 1, freq_b, hit_b, miss_b> in \(H_{II}\) through ptr and increase freq_b and hit_b by 1 and Prob_b, respectively; otherwise (wrong prediction, e.g. \(X_i = a\)), update the entry by increasing miss_b by Prob_b, retrieve the entry of \(H_{II}\) <Hash\(\"aca\"\), 1, freq_a, hit_a, miss_a> (insert a new one if it does not exist) and increment freq_a by 1 (step 3(b)). When all the leaf nodes of FI have been processed and aggregates in \(H_{II}\) updated, we recycle all the nodes in FI and start a new round to predict values in the next \(\Delta\) timestamps. Figure 5.2 illustrates the pseudo code of the probabilistic algorithm.

**Algorithm Prob_Prediction**

// step 1(a): prediction
1. for each time series \(T_t[0 : m - 1]\) (\(1 \leq i \leq n\))
2. for each subsequence \(T_t[(m - 1)-j+1:m-1+j]\) (\(1 \leq j \leq \Delta\))
3. convert it into symbolic representation \(s_1s_2\ldots s_{h-1}X\)
   // X is the symbol to predict
4. transform \(s_1\), \(s_2\), ..., and \(s_{h-1}\) to integers \(s_1', s_2', \ldots, \text{and} \ s_{h-1}'\)
5. search \(H_{II}\) with keys \(\langle s_1', s_2', \ldots, s_{h-1}'\rangle, i\)>
6. retrieve all entries in the form <Hash\(\"s_1s_2\ldots s_{h-1}X'\"\), i, freq_X', hit_X', miss_X'>
7. predict X as \(X'\) with probability Prob_X'
   \(<\text{freq}_X', \text{hit}_X', \text{miss}_X'>\)
// step 1(b): construct the index for similarity search
8. insert entry <Hash\(\"X_{h-1}X'\"\), i, \(m + j\), ptr>
   into FI with the key Hash\(\"s_1s_2\ldots s_{h-1}X'\"\)
   where ptr points to <Hash\(\"s_1s_2\ldots s_{h-1}X'\"\), i, freq_X', hit_X', miss_X'>
// step 2: answer similarity search query on FI
1. Similarity_search (FI, Q, L)
// step 3(a): after \(\Delta\) timestamps, batch update
1. batch scan all the leaf nodes of FI
2. for each entry <Hash\(\"X_{h-1}X'\"\), i, \(m - j\), ptr>
   // step 3(b): update statistics
3. obtain the true value \(X''\) of X in \(T_t\)
   with end offset (m-1+j)
4. obtain entry <Hash\(\"s_1s_2\ldots s_{h-1}X''\"\), i, freq_X'', hit_X'', miss_X''> in \(H_{II}\) through ptr
5. if \(X'' = X''\)
6. freq_X' = freq_X' + 1 and hit_X'' = hit_X'' + Prob_X'
7. else miss_X'' = miss_X'' + Prob_X'
8. retrieve entry <Hash\(\"s_1s_2\ldots s_{h-1}X''\"\), i, freq_X'', hit_X'', miss_X''>
9. set freq_X'' = freq_X'' + 1
10. recycle all the node in \(H_{II}\) and repeat step 1
   predicting the next \(\Delta\) values

**End Prob_Prediction**

![Figure 5.1: Example of Processes of Predictions, Queries and Updates](Image)

![Figure 5.2: The Pseudo Code of Probabilistic Prediction](Image)
6. Experimental Evaluation

In order to evaluate the effectiveness and efficiency of our proposed approaches, we run extensive experiments with both synthetic and real datasets. Specifically, synthetic datasets include the periodic dataset \(^1\) (100K) and randomwalk dataset (5120 sequences of length 512) \([17, 18, 35]\), whereas real datasets consist of 24 benchmark datasets (200 sequences of length 256 for each dataset) \([18, 35, 8]\) and sstock dataset (5120 sequences of length 512) \([30]\). In particular, the real datasets, 24 benchmark and sstock datasets, represent a wide spectrum of applications and data characteristics, which can be used to verify the universal usefulness of our approaches. Furthermore, when we perform our predictions, we always assume that the underlying data models are not known in advance. In order to verify the effectiveness, we compare the probabilistic prediction with both linear and quadratic predictions, in terms of prediction and query accuracy, under different parameters, that is, \(K, l, H, \Delta t\) and \(e\). Section 6.1 demonstrates the experiment settings and tunes parameters \(K\) and \(l\) for the following experiments. Section 6.2 evaluates the prediction accuracy and efficiency of our probabilistic approach compared to that of linear, quadratic, and fuzzy solutions. Section 6.3 presents the query accuracy of similarity search with the predicted future subsequences. In Section 6.4, we present results of efficiency test of the similarity search.

6.1. Experiment Settings and Parameter Tuning

The first set of experiments study the impact of parameters \(K\) and \(l\) on the prediction accuracy of our probabilistic approach, and then fix their values for all the subsequent experiments. Note that, however, each predicted value in the proposed solution is not a single value, but a range corresponding to a symbol, for example, \([\text{bound}_{l,1}, \text{bound}_{l,2}]\). Let \(act\) be the actual value. We define two additional types of prediction error specific to the probabilistic approach, that is, the minimum error \(\text{Err}_{\text{prob}, \text{min}}\) and average error \(\text{Err}_{\text{prob}, \text{avg}}\) defined as follows:

\[
\text{Err}_{\text{prob}, \text{min}}(\text{act}) = \begin{cases} 
(\text{bound}_{l,2} - \text{act})^2 & \text{if } \text{bound}_{l,2} \geq \text{act} \\
(\text{bound}_{l,1} - \text{act})^2 & \text{if } \text{bound}_{l,1} \leq \text{act} \\
0 & \text{otherwise}
\end{cases}
\]

\[
\text{Err}_{\text{prob}, \text{avg}}(\text{act}) = \frac{(\text{bound}_{l,2} + \text{bound}_{l,1})/2 - \text{act}^2}{2}
\]

Intuitively, as long as the actual value falls into the predicted range, the minimum error treats it as zero error. On the other hand, the average error always selects the middle value of the range, that is, \((\text{bound}_{l,2} + \text{bound}_{l,1})/2\), as the estimated value.

Figure 6.1(a) illustrates the effect of parameter \(K\) (= 8, 16, 32, 64, 128), that is, the total number of symbols in the alphabet table, on the prediction accuracy with periodic dataset, where \(l = 128, H = 512\) and \(\Delta t = 256\).

When \(K\) is small, that is, the value range of the time series is divided into only a few ranges, errors of the probabilistic method are high, since the granularity of the discretization is too coarse to give precise result. When \(K\) is too large, however, errors are also high. This is mainly because aggregates in the hash index may not be well trained. Figure 6.1(b) shows the prediction errors with different \(l\) on periodic dataset, that is, 64, 96, 128, 256, and 512, where \(H = 512\) and \(\Delta t = 256\). Since large \(l\) indicates the approximation with coarse resolution, the prediction error increases dramatically with \(l\). We did the same set of experiments on other datasets and the results are similar. Thus, in the sequel, we will fix the value of \(K\) to 32 and choose \(l = 128\).

6.2. Prediction Accuracy and Efficiency

After fixing values of \(K\) and \(l\), we consider the effect of \(H\) and \(\Delta t\) on prediction accuracy of four methods, linear, quadratic, and probabilistic. In particular, the prediction accuracy of linear and quadratic methods is measured by the prediction error \(\text{Err}_{\text{pred}}\) defined in Formula (2), while that of the probabilistic approach by the minimum error \(\text{Err}_{\text{prob}, \text{min}}\) and average error \(\text{Err}_{\text{prob}, \text{avg}}\) in Formulas (8) and (9), respectively. In Figure 6.2(a), we fix \(\Delta t = 256\) and vary \(H\) = 256, 384, 512, 640 and 768. Both linear and quadratic predictions have higher errors when \(H\) increases. This indicates the most recent values have more importance in predicting future values with these two methods. Oppositely, however, the probabilistic approach gives better result when \(H\) increases. Figure 6.2(b) varies \(\Delta t\) from 64 to 512, where \(H = 512\). Since greater \(\Delta t\) can give larger errors, the y-axis in the figure is defined as \(\text{Err}_{\text{pred}} / \Delta t\). All the three methods have higher errors when \(\Delta t\) is large. Both linear and quadratic approaches show nice ability of predicting short term values. However, their errors dramatically increase with \(\Delta t\), whereas the probabilistic one does not increase so quickly.

---

\(^1\) http://kdd.ics.uci.edu\/summary.data.type.html
In order to further validate the effectiveness of our methods, we run the same set of experiments on 24 benchmark, sstock and randomwalk datasets. In particular, 24 benchmark consists of 24 real datasets, each containing 200 time series of length 256, where we set $\Delta t = 256$ and $H = 512$. The sstock dataset is a real stock data that contains 193 company stocks’ daily closing price from late 1993 to early 1996, each consisting of 512 values. The randomwalk dataset is synthetic containing time series of length 512. For both sstock and randomwalk, we let $\Delta t = 512$ and $H = 1024$. Furthermore, we also compare our proposed approaches with the fuzzy method [19]. The experimental results are depicted in Figure 6.3, which shows better performance of the probabilistic solution than that of linear, quadratic and fuzzy. Note that, for the reason of clear illustrations, we sort 24 benchmark datasets in the chart by their errors of linear prediction. The prediction error of fuzzy method is not very stable, but always greater than that of the probabilistic one. Moreover, as illustrated in Figure 6.4, the CPU time of linear and quadratic is the lowest, whereas that of probabilistic is about $10^3$ ms, and fuzzy orders of magnitude higher, which indicates its inability in the stream processing.

Figure 6.3: Prediction Accuracy vs. Dataset

Figure 6.4: Prediction Efficiency vs. Dataset

6.3. Query Accuracy

Next, we demonstrate the query accuracy of the similarity search in future time series. Specifically, we run our experiments on two datasets, sstock and randomwalk, respectively. We divide each dataset into 128 time series, that is, $n = 128$. Consistent to previous settings, we choose $H = 1024, \Delta t = 512$ and select $L = 384$ as the query length ($1024 \geq 512 + 384$). Therefore, during each blocked period, there are totally $n \Delta t = (128 \times 512)$ predicted values (future subsequences of length 384). We randomly extract 128 subsequences of length 384 from each dataset and use them as our query sequences. Figure 6.5 illustrates the impact of the similarity threshold $\epsilon$ on the query accuracy, in terms of recall ratio defined in Formula (3). Here, the query accuracy mainly depends on the prediction accuracy. Furthermore, since people are usually interested in finding future series that follow certain patterns with false dismissals as low as possible, our major concern is high recall ratio. Interestingly, for both datasets, the recall ratio of the probabilistic approach is much better than those of linear and quadratic predictions. When the value of $\epsilon$ increases, the query result contains more candidates. Therefore, the recall ratio appears increasing for all the three approaches (the last half of the curve in probabilistic one). When $\epsilon$ is very small, however, both linear and quadratic have very few candidates, that is, close to zero candidate on average, thus leading to near zero recall ratio. On the other hand, the query result of the probabilistic approach contains more candidates with small $\epsilon$, thus having higher recall ratio.

Figure 6.5: Query Accuracy vs. $\epsilon$

6.4. Query Efficiency

Finally, we evaluate the query efficiency of the similarity search in terms of the I/O cost. Specifically, we compare the similarity search on B+-tree index FI with that of the sequential scan. We set the page size to 1024 bytes, and run the experiments on datasets sstock and randomwalk, respectively, where $n = 128, H = 1024, \Delta t = 512$ and $L = 384$. Similar to the previous experiment settings, we extract 128 query sequences of length 384. The value of $\epsilon$ is chosen so that the selectivity of the query result is about 0.1%. Figure 6.6 illustrates the average number of page accesses at different query timestamps. For the sequential scan, since each future subsequence takes up 5 bytes (8 symbols with five bits each), each page of length 1024 contains 204 subsequences. Therefore, the page accesses of the sequential scan are always $n \Delta t / 200$, that is, $320$. On the other hand, the I/O cost of our FI index is around 50 with both datasets. In summary, the probabilistic approach works well on the prediction.
Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM'06)

0-7695-2590-3/06 $20.00 © 2006 IEEE

accuracy and also achieves better query accuracy and efficiency than that of the polynomial approach and sequential scan, respectively.

7. Conclusions
In this paper, we address the issues of similarity search over future stream time series. In many applications, data are often delayed or arrive in batches. In order to offer a reasonably accurate answer, we need methods to predict values and carry out similarity searches over these predicted values. We propose two prediction techniques, polynomial and probabilistic, to predict future values in the stream environment. We also present efficient indexes to facilitate the probabilistic approach with both prediction tasks and similarity searches in future time series. In this paper, we currently consider the problem with a fixed length of the blocked period, and it would be interesting to explore scenarios in which the period length is not known in advance. Moreover, although the previous fuzzy or data mining approaches are not suitable for stream processing, they can achieve prediction accuracy with an error bound. Therefore, modeling our approaches with proven error bounds constitutes an interesting problem.

Acknowledgements
Funding for this work was provided by the Hong Kong RGC grants DAG05/06.EG03.

8. References


[27] Vilalta, R., Ma, S. Predicting rare events in temporal domains. ICDM, 2002.


